Lower bounds for polynomial kernelization

Michał Pilipczuk

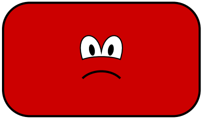
Institute of Informatics, University of Warsaw, Poland

Parameterized Complexity Summer School

Vienna, September 2$^{\mathrm{nd}}$, 2017

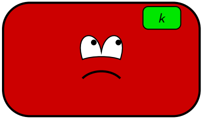
Kernelization — recap

Kernelization — recap



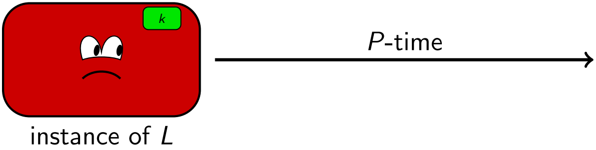
instance of L

Kernelization — recap

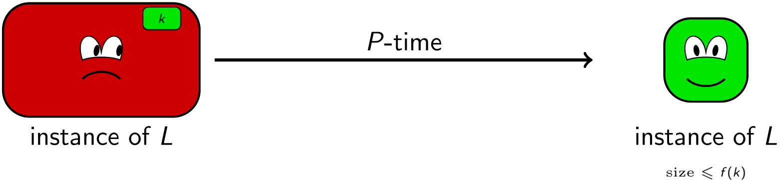


instance of L

Kernelization — recap



Kernelization — recap



Background in complexity theory

Unparameterized problems

$\Leftrightarrow$

Languages over Σ, for a finite alphabet Σ

$\Leftrightarrow$

Subsets of $\Sigma^\star$

Background in complexity theory

Unparameterized problems

$\Leftrightarrow$

Languages over Σ, for a finite alphabet Σ

$\Leftrightarrow$

Subsets of $\Sigma^\star$

Parameterized problems

$\Leftrightarrow$

Sets of pairs (x, k), where $x \in \Sigma^\star$ and k is a nonnegative integer

Background in complexity theory

Unparameterized problems

$\Leftrightarrow$

Languages over Σ, for a finite alphabet Σ

$\Leftrightarrow$

Subsets of $\Sigma^\star$

Parameterized problems

$\Leftrightarrow$

Sets of pairs (x, k), where $x \in \Sigma^\star$ and k is a nonnegative integer

- **Unparameterized variant**: k is appended to x in unary.

Background in complexity theory

Unparameterized problems

$\Leftrightarrow$

Languages over Σ, for a finite alphabet Σ

$\Leftrightarrow$

Subsets of $\Sigma^\star$

Parameterized problems

$\Leftrightarrow$

Sets of pairs (x, k), where $x \in \Sigma^\star$ and k is a nonnegative integer

- **Unparameterized variant**: k is appended to x in unary.
- **Kernelization algorithm** takes on input an instance (x, k), and outputs an instance (x', k') such that

$$(x, k) \in L \Leftrightarrow (x', k') \in L \qquad \text{and} \qquad |x'| + k' \leqslant f(k)$$

for some computable function f.

Kernelization and FPT

- If a decidable problem has a kernelization algorithm, then it is FPT.

Kernelization and FPT

- If a decidable problem has a kernelization algorithm, then it is FPT.
- Any FPT problem admits a kernelization algorithm:

Kernelization and FPT

- If a decidable problem has a kernelization algorithm, then it is FPT.
- Any FPT problem admits a kernelization algorithm:
  - Let (x, k) be the input instance.

Kernelization and FPT

- If a decidable problem has a kernelization algorithm, then it is FPT.
- Any FPT problem admits a kernelization algorithm:
  - Let (x, k) be the input instance.
  - If $|x| \leqslant f(k)$, then we already have a kernel.

Kernelization and FPT

- If a decidable problem has a kernelization algorithm, then it is FPT.
- Any FPT problem admits a kernelization algorithm:
  - Let (x, k) be the input instance.
  - If $|x| \leqslant f(k)$, then we already have a kernel.
  - Otherwise $f(k) \cdot |x|^c = \mathcal{O}(|x|^{c+1})$.

Kernelization and FPT

- If a decidable problem has a kernelization algorithm, then it is FPT.
- Any FPT problem admits a kernelization algorithm:
  - Let (x, k) be the input instance.
  - If $|x| \leqslant f(k)$, then we already have a kernel.
  - Otherwise $f(k) \cdot |x|^c = \mathcal{O}(|x|^{c+1})$.
- Question of existence of **any kernel** is equivalent to being FPT.

Kernelization and FPT

- If a decidable problem has a kernelization algorithm, then it is FPT.
- Any FPT problem admits a kernelization algorithm:
  - Let (x, k) be the input instance.
  - If $|x| \leqslant f(k)$, then we already have a kernel.
  - Otherwise $f(k) \cdot |x|^c = \mathcal{O}(|x|^{c+1})$.
- Question of existence of **any kernel** is equivalent to being FPT.
- We are interested in **polynomial kernels**, where f is a polynomial.

Kernelization and FPT

- If a decidable problem has a kernelization algorithm, then it is FPT.
- Any FPT problem admits a kernelization algorithm:
  - Let (x, k) be the input instance.
  - If $|x| \leqslant f(k)$, then we already have a kernel.
  - Otherwise $f(k) \cdot |x|^c = \mathcal{O}(|x|^{c+1})$.
- Question of existence of **any kernel** is equivalent to being FPT.
- We are interested in **polynomial kernels**, where f is a polynomial.
- Before 2008, no tool to classify FPT problems wrt. whether they have polykernels or not.

Motivating intuition

- Consider the k-PATH problem: verify whether the input graph contains a simple path on k vertices.

Motivating intuition

- Consider the k-PATH problem: verify whether the input graph contains a simple path on k vertices.
- Suppose for a moment that k-PATH admits a kernelization algorithm that, say, produces kernels with at most k^3 vertices.

Motivating intuition

- Consider the k-PATH problem: verify whether the input graph contains a simple path on k vertices.
- Suppose for a moment that k-PATH admits a kernelization algorithm that, say, produces kernels with at most k^3 vertices.
- Take $t = k^7$ instances $(G_1, k), (G_2, k), \ldots, (G_t, k)$.

Motivating intuition

- Consider the k-Path problem: verify whether the input graph contains a simple path on k vertices.
- Suppose for a moment that k-Path admits a kernelization algorithm that, say, produces kernels with at most k^3 vertices.
- Take $t = k^7$ instances $(G_1, k), (G_2, k), \ldots, (G_t, k)$.
- Let H be a disjoint union of $G_1, G_2, \ldots, G_t$. Then the answer to (H, k) is YES if and only if the answer to any (G_i, k) is YES.

Motivating intuition

- Consider the k-Path problem: verify whether the input graph contains a simple path on k vertices.
- Suppose for a moment that k-Path admits a kernelization algorithm that, say, produces kernels with at most k^3 vertices.
- Take $t = k^7$ instances $(G_1, k), (G_2, k), \ldots, (G_t, k)$.
- Let H be a disjoint union of $G_1, G_2, \ldots, G_t$. Then the answer to (H, k) is YES if and only if the answer to any (G_i, k) is YES.
- Apply kernelization to (H, k) obtaining an instance with k^3 vertices, encodable in k^6 bits.

Motivating intuition

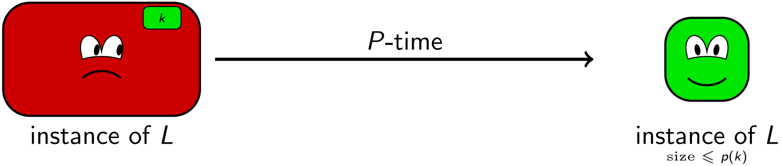
- Consider the k-PATH problem: verify whether the input graph contains a simple path on k vertices.
- Suppose for a moment that k-PATH admits a kernelization algorithm that, say, produces kernels with at most k^3 vertices.
- Take $t = k^7$ instances $(G_1, k), (G_2, k), \ldots, (G_t, k)$.
- Let H be a disjoint union of $G_1, G_2, \ldots, G_t$. Then the answer to (H, k) is YES if and only if the answer to any (G_i, k) is YES.
- Apply kernelization to (H, k) obtaining an instance with k^3 vertices, encodable in k^6 bits.

Intuition

The final number of bits is much less than the number input instances. Most of the instances have to be **discarded completely**.

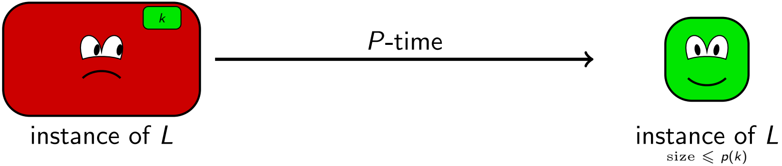
Kernelization and Compression

KERNELIZATION

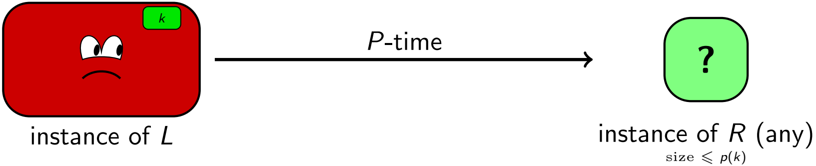


Kernelization and Compression

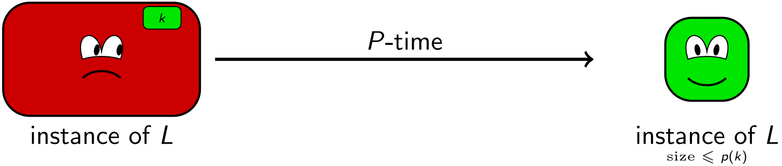
KERNELIZATION



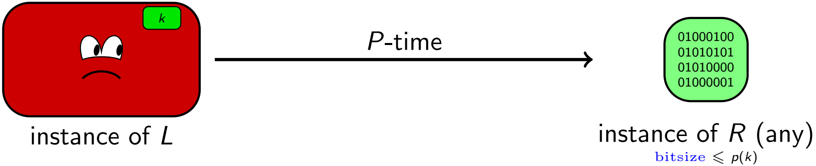
COMPRESSION



KERNELIZATION



COMPRESSION



Kernelization and Compression

- **Intuition**: In compression we only care about shrinking the size of the instance to a small size without mixing YES- and NO-instances.

Kernelization and Compression

- **Intuition**: In compression we only care about shrinking the size of the instance to a small size without mixing YES- and NO-instances.
- A polynomial kernelization is always a polynomial compression.

Kernelization and Compression

- **Intuition**: In compression we only care about shrinking the size of the instance to a small size without mixing YES- and NO-instances.
- A polynomial kernelization is always a polynomial compression.
- A polynomial compression can be turned into a polynomial kernelization provided that there is a **P**-reduction from R to L.

Kernelization and Compression

- **Intuition**: In compression we only care about shrinking the size of the instance to a small size without mixing YES- and NO-instances.
- A polynomial kernelization is always a polynomial compression.
- A polynomial compression can be turned into a polynomial kernelization provided that there is a **P**-reduction from R to L.
  - For instance, when $R \in$ **NP** and L is **NP**-hard.

Kernelization and Compression

- **Intuition**: In compression we only care about shrinking the size of the instance to a small size without mixing YES- and NO-instances.
- A polynomial kernelization is always a polynomial compression.
- A polynomial compression can be turned into a polynomial kernelization provided that there is a **P**-reduction from R to L.
  - For instance, when $R \in \mathbf{NP}$ and L is **NP**-hard.
- **Note**: There are examples when a poly-compression is known but a poly-kernel is not known, because it is unclear whether R is in **NP**.

OR-distillation

- Let L, R be *unparameterized* languages.

OR-distillation

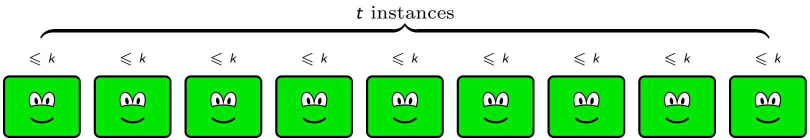
- Let L, R be *unparameterized* languages.

OR-distillation of L into R

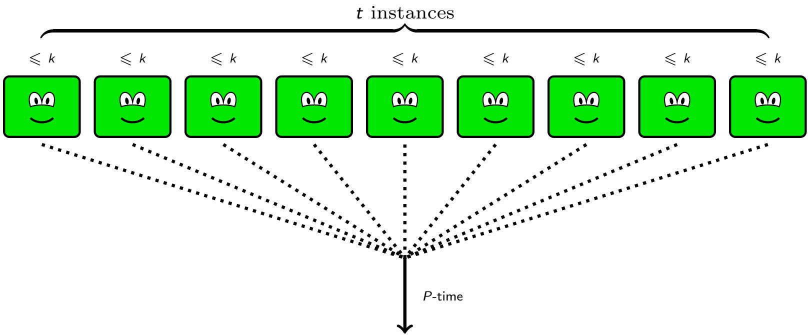
Input: Words $x_1, x_2, \ldots, x_t$, each of length at most k.
Time: $\mathrm{poly}(t + \sum_{i=1}^{t} |x_i|)$.
Output: One word y such that
(a) $|y| = \mathrm{poly}(k)$, and
(b) $y \in R$ if and only if $x_i \in L$ for at least one i.

OR-distillation on picture

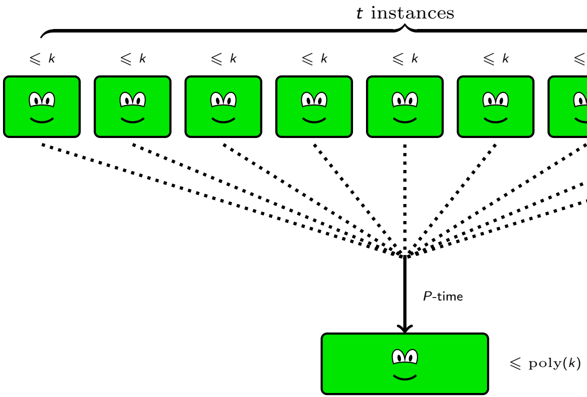
OR-distillation on picture



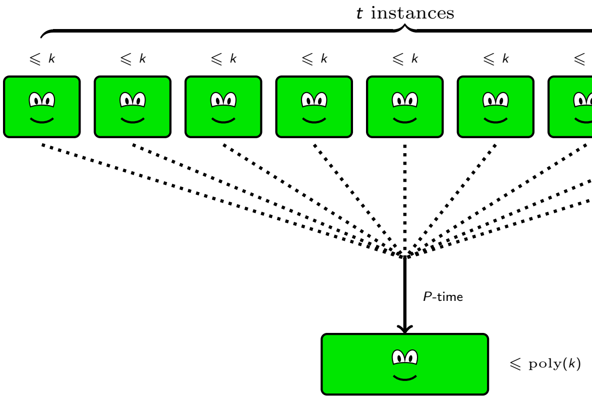
OR-distillation on picture



OR-distillation on picture

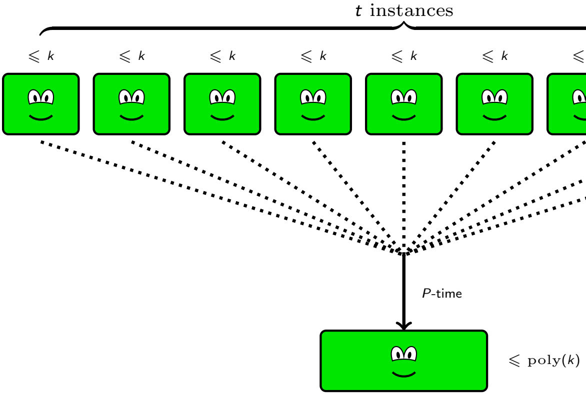


OR-distillation on picture



Intuition: Necessary loss of information $\rightsquigarrow$ Contradiction for an **NP**-hard L

OR-distillation on picture



Intuition: Necessary loss of information $\rightsquigarrow$ Contradiction for an **NP**-hard L

Define $\text{OR-}L = \{x_1 \# x_2 \# \ldots \# x_t \ : \ x_i \in L \text{ for at least one } i\}$.

OR-distillation $L \rightarrow R$ is a polynomial compression OR-$L/\max|x_i| \rightarrow R$

Backbone theorem

OR-distillation theorem [Fortnow, Santhanam; 2008]

SAT does not admit an OR-distillation algorithm into any language R, unless $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.

Backbone theorem

OR-distillation theorem [Fortnow, Santhanam; 2008]

SAT does not admit an OR-distillation algorithm into any language R, unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

Corollary

No **NP**-hard problem admits an OR-distillation algorithm into any language R, unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

Backbone theorem

OR-distillation theorem [Fortnow, Santhanam; 2008]

SAT does not admit an OR-distillation algorithm into any language R, unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

Corollary

No **NP**-hard problem admits an OR-distillation algorithm into any language R, unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

- Assumption **NP** $\subseteq$ **coNP**$/\mathrm{poly}$ may seem mysterious.

Backbone theorem

OR-distillation theorem [Fortnow, Santhanam; 2008]

SAT does not admit an OR-distillation algorithm into any language R, unless $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.

Corollary

No **NP**-hard problem admits an OR-distillation algorithm into any language R, unless $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.

- Assumption $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$ may seem mysterious.
  - **Intuition**: Verifying proofs in **P**-time cannot be turned into verifying counterexamples in **P**-time, even if we allow *polynomial advice*.

Backbone theorem

OR-distillation theorem [Fortnow, Santhanam; 2008]

SAT does not admit an OR-distillation algorithm into any language R, unless $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.

Corollary

No **NP**-hard problem admits an OR-distillation algorithm into any language R, unless $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.

- Assumption $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$ may seem mysterious.
  - **Intuition**: Verifying proofs in **P**-time cannot be turned into verifying counterexamples in **P**-time, even if we allow *polynomial advice*.
  - $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$ implies $\mathrm{PH} = \Sigma_3^{\mathrm{P}}$.

Backbone theorem

OR-distillation theorem [Fortnow, Santhanam; 2008]

SAT does not admit an OR-distillation algorithm into any language R, unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

Corollary

No **NP**-hard problem admits an OR-distillation algorithm into any language R, unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

- Assumption **NP** $\subseteq$ **coNP**$/\mathrm{poly}$ may seem mysterious.
  - **Intuition**: Verifying proofs in **P**-time cannot be turned into verifying counterexamples in **P**-time, even if we allow *polynomial advice*.
  - **NP** $\subseteq$ **coNP**$/\mathrm{poly}$ implies $\mathrm{PH} = \Sigma_3^{\mathrm{P}}$.
  - Not as bad as **P** $=$ **NP**, but still considered very unlikely.

Backbone theorem

OR-distillation theorem [Fortnow, Santhanam; 2008]

SAT does not admit an OR-distillation algorithm into any language R, unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

Corollary

No **NP**-hard problem admits an OR-distillation algorithm into any language R, unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

- Assumption **NP** $\subseteq$ **coNP**$/\mathrm{poly}$ may seem mysterious.
  - **Intuition**: Verifying proofs in **P**-time cannot be turned into verifying counterexamples in **P**-time, even if we allow *polynomial advice*.
  - **NP** $\subseteq$ **coNP**$/\mathrm{poly}$ implies $\mathrm{PH} = \Sigma_3^{\mathrm{P}}$.
  - Not as bad as **P** $=$ **NP**, but still considered very unlikely.
- The proof is very short, but very tricky.

OR-composition

- Let L be a *parameterized* language.

OR-composition

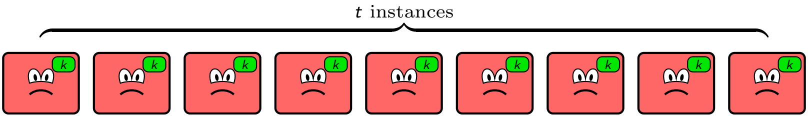
- Let L be a *parameterized* language.

OR-composition algorithm for L

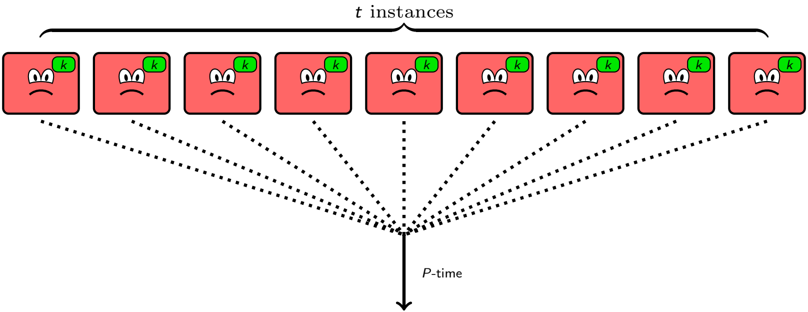
Input: Instances $(x_1, k), (x_2, k), \ldots, (x_t, k)$.
Time: $\mathrm{poly}(t + \sum_{i=1}^{t} |x_i| + k)$.
Output: One instance $(y, k^\star)$ such that
(a) $k^\star = \mathrm{poly}(k)$, and
(b) $(y, k^\star) \in L$ iff $(x_i, k) \in L$ for at least one i.

OR-composition on picture

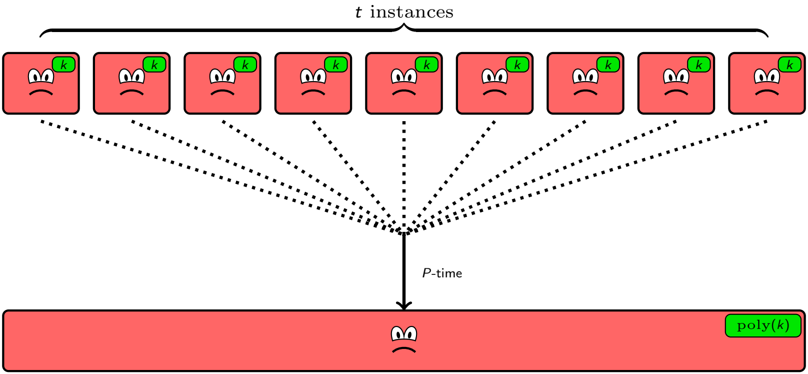
OR-composition on picture



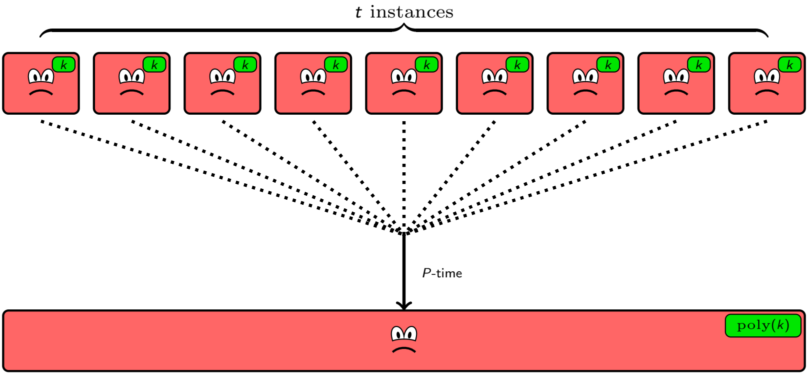
OR-composition on picture



OR-composition on picture



OR-composition on picture



OR-composition theorem [BDFH; 2008]

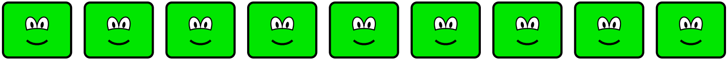
Suppose a parameterized problem L admits an OR-composition algorithm, and the unparameterized version of L is **NP**-hard.

Then L does not admit a polynomial kernel unless **NP** $\subseteq$ **coNP**/poly.

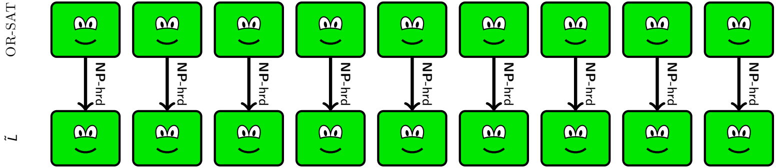
Proof

Proof

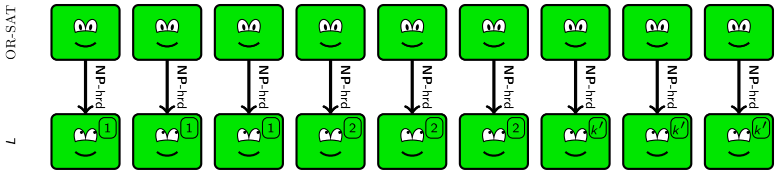
OR-SAT



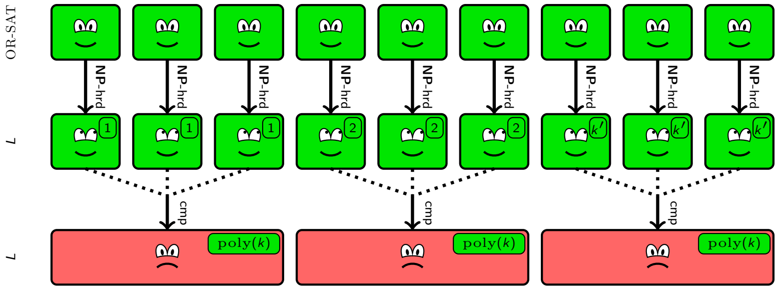
Proof



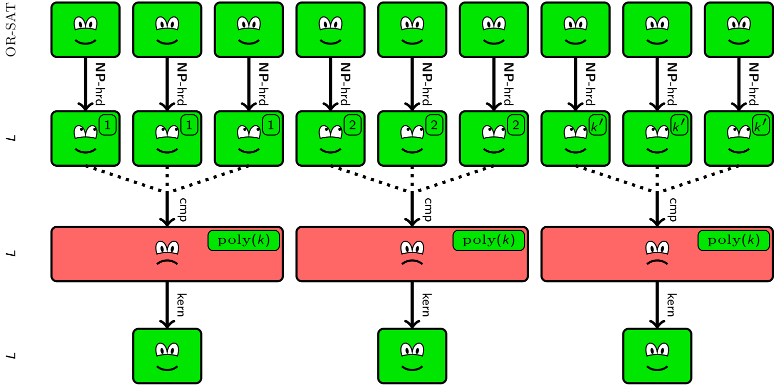
Proof



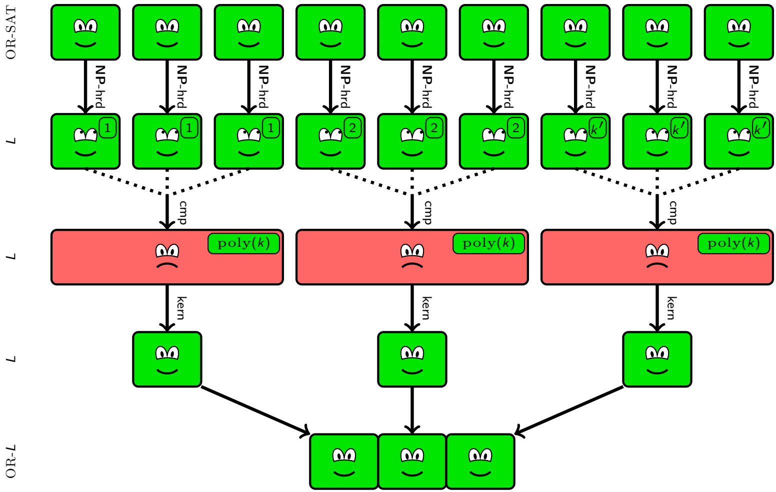
Proof



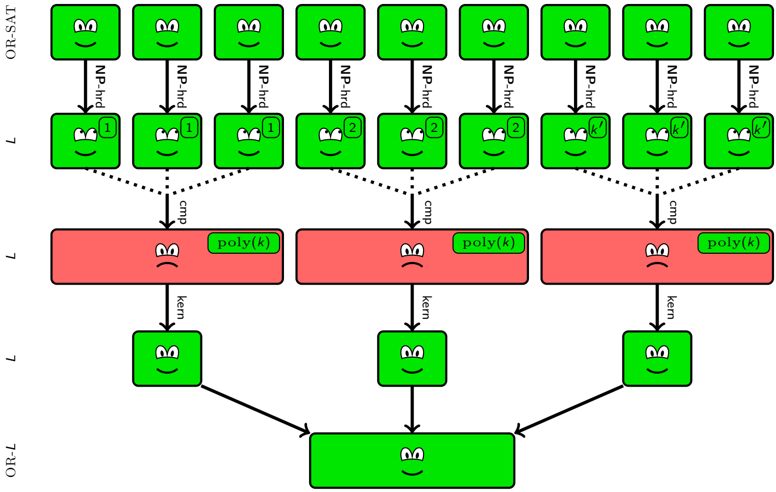
Proof



Proof



Proof



Corollaries

- k-Path does not admit a polykernel, unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

Corollaries

- k-PATH does not admit a polykernel, unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.
- **Composition**:
  Take the disjoint union of the input graphs and the same parameter.

Corollaries

- k-Path does not admit a polykernel, unless $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.
- **Composition**:
  Take the disjoint union of the input graphs and the same parameter.
  - A graph admits a k-path iff any of its connected components does.

Corollaries

- k-Path does not admit a polykernel, unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.
- **Composition**:
  Take the disjoint union of the input graphs and the same parameter.
  - A graph admits a k-path iff any of its connected components does.
- Same for k-Cycle and many other problems.

Corollaries

- k-Path does not admit a polykernel, unless $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.
- **Composition**:
  Take the disjoint union of the input graphs and the same parameter.
  - A graph admits a k-path iff any of its connected components does.
- Same for k-Cycle and many other problems.
- Today, investigating the existence of a polynomial kernel is often a secondary goal after showing that a problem is FPT.

Adding features

- Does the proof actually exclude even polynomial compression into any R, not just kernelization?

Adding features

- Does the proof actually exclude even polynomial compression into any R, not just kernelization?
  - Sure, we will just end up with an instance of OR-R.

Adding features

- Does the proof actually exclude even polynomial compression into any R, not just kernelization?
  - Sure, we will just end up with an instance of OR-R.
- Do we need to start the composition with the same language L as we apply the compression to?

Adding features

- Does the proof actually exclude even polynomial compression into any R, not just kernelization?
  - Sure, we will just end up with an instance of OR-R.
- Do we need to start the composition with the same language L as we apply the compression to?
  - No, the composition algorithm can compose instances of any **NP**-hard language Q into one instance of L.

Adding features

- Does the proof actually exclude even polynomial compression into any R, not just kernelization?
  - Sure, we will just end up with an instance of OR-R.
- Do we need to start the composition with the same language L as we apply the compression to?
  - No, the composition algorithm can compose instances of any **NP**-hard language Q into one instance of L.
- Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?

Adding features

- Does the proof actually exclude even polynomial compression into any R, not just kernelization?
  - Sure, we will just end up with an instance of OR-R.
- Do we need to start the composition with the same language L as we apply the compression to?
  - No, the composition algorithm can compose instances of any **NP**-hard language Q into one instance of L.
- Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
  - Yes, as long as we have polynomial number of buckets.

Adding features

- Does the proof actually exclude even polynomial compression into any R, not just kernelization?
  - Sure, we will just end up with an instance of OR-R.
- Do we need to start the composition with the same language L as we apply the compression to?
  - No, the composition algorithm can compose instances of any **NP**-hard language Q into one instance of L.
- Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
  - Yes, as long as we have polynomial number of buckets.
- How large can t be?

Adding features

- Does the proof actually exclude even polynomial compression into any R, not just kernelization?
  - Sure, we will just end up with an instance of OR-R.
- Do we need to start the composition with the same language L as we apply the compression to?
  - No, the composition algorithm can compose instances of any **NP**-hard language Q into one instance of L.
- Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
  - Yes, as long as we have polynomial number of buckets.
- How large can t be?
  - Well, not larger than $|\Sigma|^{k+1}$, as we may remove duplicates of the input instances.

Adding features

- Does the proof actually exclude even polynomial compression into any R, not just kernelization?
  - Sure, we will just end up with an instance of OR-R.
- Do we need to start the composition with the same language L as we apply the compression to?
  - No, the composition algorithm can compose instances of any **NP**-hard language Q into one instance of L.
- Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
  - Yes, as long as we have polynomial number of buckets.
- How large can t be?
  - Well, not larger than $|\Sigma|^{k+1}$, as we may remove duplicates of the input instances.
  - Hence, we may assume that $\log t = \mathcal{O}(k)$.

Adding features

- Does the proof actually exclude even polynomial compression into any R, not just kernelization?
  - Sure, we will just end up with an instance of OR-R.
- Do we need to start the composition with the same language L as we apply the compression to?
  - No, the composition algorithm can compose instances of any **NP**-hard language Q into one instance of L.
- Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
  - Yes, as long as we have polynomial number of buckets.
- How large can t be?
  - Well, not larger than $|\Sigma|^{k+1}$, as we may remove duplicates of the input instances.
  - Hence, we may assume that $\log t = \mathcal{O}(k)$.
  - Ergo, the parameter of the composed instance may depend polynomially on **both** k and $\log t$.

Adding features

- Does the proof actually exclude even polynomial compression into any R, not just kernelization?
  - Sure, we will just end up with an instance of OR-R.
- Do we need to start the composition with the same language L as we apply the compression to?
  - No, the composition algorithm can compose instances of any **NP**-hard language Q into one instance of L.
- Can we add more refined bucket sorting? For instance, also by the number of vertices in the graph?
  - Yes, as long as we have polynomial number of buckets.
- How large can t be?
  - Well, not larger than $|\Sigma|^{k+1}$, as we may remove duplicates of the input instances.
  - Hence, we may assume that $\log t = \mathcal{O}(k)$.
  - Ergo, the parameter of the composed instance may depend polynomially on **both** k and $\log t$.
  - Observed also earlier via different arguments. (Dom, Lokshtanov, and Saurabh; 2009)

Towards a unified methodology

- After the invention of the technique of OR-compositions, there was a huge number of no-polykernel results.

Towards a unified methodology

- After the invention of the technique of OR-compositions, there was a huge number of no-polykernel results.
  - As we'll see later, there can be much more intricate compositions than just "disjoint union".

Towards a unified methodology

- After the invention of the technique of OR-compositions, there was a huge number of no-polykernel results.
  - As we'll see later, there can be much more intricate compositions than just "disjoint union".
  - **Examples**: Max Leaf Subtree, Set Cover/m, Set Cover/n, Steiner Tree, Connected Vertex Cover, Disjoint Paths, Directed Multiway Cut with 2 terminals, ...

Towards a unified methodology

- After the invention of the technique of OR-compositions, there was a huge number of no-polykernel results.
  - As we'll see later, there can be much more intricate compositions than just "disjoint union".
  - **Examples**: Max Leaf Subtree, Set Cover/m, Set Cover/n, Steiner Tree, Connected Vertex Cover, Disjoint Paths, Directed Multiway Cut with 2 terminals, ...
- Most of the works use a subset of mentioned features.

Towards a unified methodology

- After the invention of the technique of OR-compositions, there was a huge number of no-polykernel results.
  - As we'll see later, there can be much more intricate compositions than just "disjoint union".
  - **Examples**: Max Leaf Subtree, Set Cover/m, Set Cover/n, Steiner Tree, Connected Vertex Cover, Disjoint Paths, Directed Multiway Cut with 2 terminals, ...
- Most of the works use a subset of mentioned features.
- **Later**: a new formalism **cross-composition** gathers all the features. (Bodlaender, Jansen, and Kratsch; 2011)

Polynomial equivalence relation

Polynomial equivalence relation

Equivalence relation $\sim$ on $\Sigma^\star$ is a **polynomial equivalence relation** if:

- checking whether two words $x, y \in \Sigma^\star$ are $\sim$-equivalent can be done in $\mathrm{poly}(|x| + |y|)$ time; and
- $\sim$ partitions words of length $\leqslant n$ into $\mathrm{poly}(n)$ equivalence classes.

Polynomial equivalence relation

Polynomial equivalence relation

Equivalence relation $\sim$ on $\Sigma^\star$ is a **polynomial equivalence relation** if:

- checking whether two words $x, y \in \Sigma^\star$ are $\sim$-equivalent can be done in $\mathrm{poly}(|x| + |y|)$ time; and
- $\sim$ partitions words of length $\leqslant n$ into $\mathrm{poly}(n)$ equivalence classes.

Polynomial equivalence relation

Polynomial equivalence relation

Equivalence relation $\sim$ on $\Sigma^\star$ is a **polynomial equivalence relation** if:

- checking whether two words $x, y \in \Sigma^\star$ are $\sim$-equivalent can be done in $\mathrm{poly}(|x| + |y|)$ time; and
- $\sim$ partitions words of length $\leqslant n$ into $\mathrm{poly}(n)$ equivalence classes.

- **Examples**, supposing some reasonable graph encoding:

Polynomial equivalence relation

Polynomial equivalence relation

Equivalence relation $\sim$ on $\Sigma^\star$ is a **polynomial equivalence relation** if:

- checking whether two words $x, y \in \Sigma^\star$ are $\sim$-equivalent can be done in $\mathrm{poly}(|x| + |y|)$ time; and
- $\sim$ partitions words of length $\leqslant n$ into $\mathrm{poly}(n)$ equivalence classes.

- **Examples**, supposing some reasonable graph encoding:
  - partitioning with respect to the number of vertices of the graph;

Polynomial equivalence relation

Polynomial equivalence relation

Equivalence relation $\sim$ on $\Sigma^\star$ is a **polynomial equivalence relation** if:

- checking whether two words $x, y \in \Sigma^\star$ are $\sim$-equivalent can be done in $\mathrm{poly}(|x| + |y|)$ time; and
- $\sim$ partitions words of length $\leqslant n$ into $\mathrm{poly}(n)$ equivalence classes.

- **Examples**, supposing some reasonable graph encoding:
  - partitioning with respect to the number of vertices of the graph;
  - or with respect to *(i)* the number of vertices, *(ii)* the number of edges, *(iii)* size of the maximum matching, *(iv)* budget.

Cross-composition

Cross-composition

An unparameterized problem Q **cross-composes** into a parameterized problem L, if there exists a polynomial equivalence relation $\sim$ and an algorithm that, given $\sim$-equivalent strings $x_1, x_2, \ldots, x_t$, in time $\mathrm{poly}\left(t + \sum_{i=1}^{t} |x_i|\right)$ produces one instance $(y, k^\star)$ such that

- $(y, k^\star) \in L$ iff $x_i \in Q$ for at least one $i = 1, 2, \ldots, t$,
- $k^\star = \mathrm{poly}\left(\log t + \max_{i=1}^{t} |x_i|\right)$.

Cross-composition

Cross-composition

An unparameterized problem Q **cross-composes** into a parameterized problem L, if there exists a polynomial equivalence relation $\sim$ and an algorithm that, given $\sim$-equivalent strings $x_1, x_2, \ldots, x_t$, in time $\mathrm{poly}\left(t + \sum_{i=1}^{t} |x_i|\right)$ produces one instance $(y, k^\star)$ such that

- $(y, k^\star) \in L$ iff $x_i \in Q$ for at least one $i = 1, 2, \ldots, t$,
- $k^\star = \mathrm{poly}\left(\log t + \max_{i=1}^{t} |x_i|\right)$.

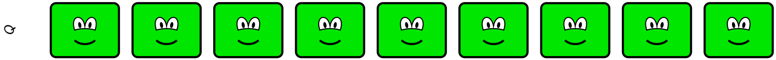
Cross-composition theorem [Bodlaender, Jansen, Kratsch]

If some **NP**-hard problem Q cross-composes into L, then L has no polynomial compression into any language R, unless **NP** $\subseteq$ **coNP**/poly.

Proof

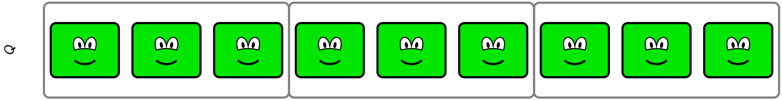
Proof

$k = \max |x_i|, \quad \log t = \mathcal{O}(k)$



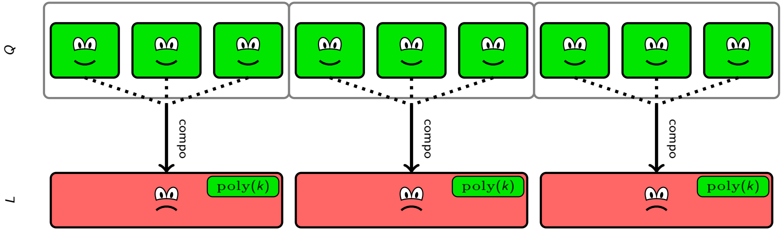
Proof

$k = \max |x_i|, \quad \log t = \mathcal{O}(k)$



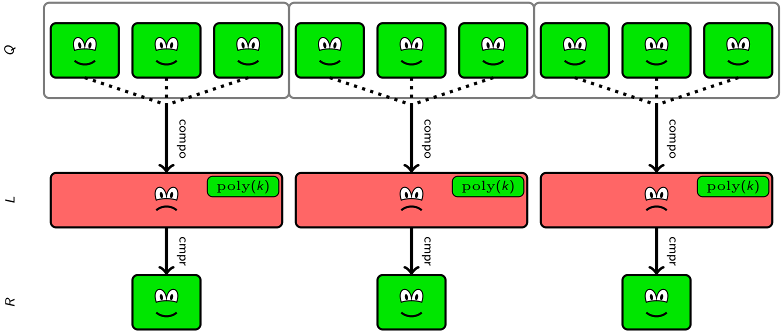
Proof

$k = \max |x_i|, \quad \log t = \mathcal{O}(k)$



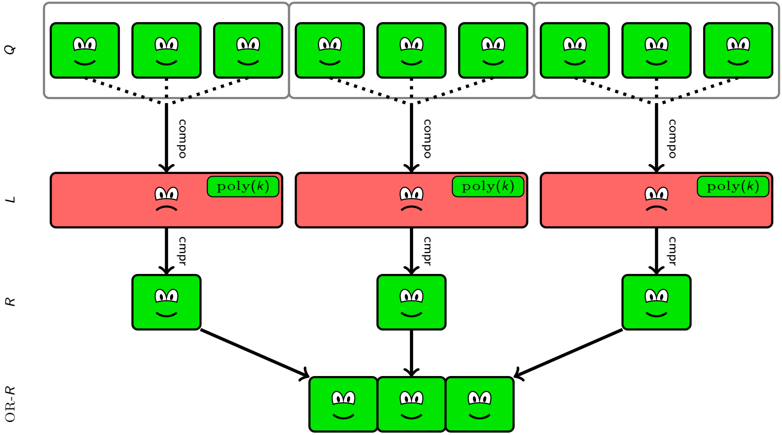
Proof

$k = \max |x_i|, \quad \log t = \mathcal{O}(k)$



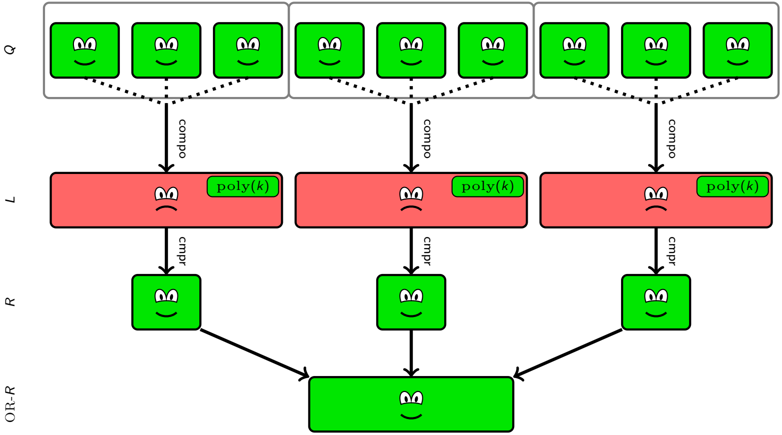
Proof

$k = \max |x_i|, \quad \log t = \mathcal{O}(k)$



Proof

$k = \max |x_i|, \quad \log t = \mathcal{O}(k)$



Applications

- Original application of Bodlaender, Jansen and Kratsch was that of **structural parameters**.

Applications

- Original application of Bodlaender, Jansen and Kratsch was that of **structural parameters**.
- In fact, cross-composition is a good framework to express also all the previous results.

Applications

- Original application of Bodlaender, Jansen and Kratsch was that of **structural parameters**.
- In fact, cross-composition is a good framework to express also all the previous results.
- **Plan for now**: show some non-trivial cross-composition to give an intuition about basic tricks.

Application 1: Set Splitting

Set Splitting

I: Universe U and family of subsets $\mathcal{F} \subseteq 2^U$

P: $|U|$

Q: Is there a coloring $\mathcal{C} : U \to \{\mathbf{B}, \mathbf{W}\}$ such that every set $X \in \mathcal{F}$ is split, i.e., contains a black and a white element?

Application 1: Set Splitting

Set Splitting

I: Universe U and family of subsets $\mathcal{F} \subseteq 2^U$
P: $|U|$
Q: Is there a coloring $\mathcal{C} : U \to \{\mathbf{B}, \mathbf{W}\}$ such that every set $X \in \mathcal{F}$ is split, i.e., contains a black and a white element?

- We show a cross-composition of Set Splitting into itself.

Application 1: Set Splitting

Set Splitting

I: Universe U and family of subsets $\mathcal{F} \subseteq 2^U$
P: $|U|$
Q: Is there a coloring $\mathcal{C} : U \to \{\mathbf{B}, \mathbf{W}\}$ such that every set $X \in \mathcal{F}$ is split, i.e., contains a black and a white element?

- We show a cross-composition of Set Splitting into itself.
- We may assume that the universes are of the same size, hence we think of them as of one, common universe.

Application 1: Set Splitting

Set Splitting

I:	Universe U and family of subsets $\mathcal{F} \subseteq 2^U$
P:	$\|U\|$
Q:	Is there a coloring $\mathcal{C} : U \to \{\mathbf{B}, \mathbf{W}\}$ such that every set $X \in \mathcal{F}$ is split, i.e., contains a black and a white element?

- We show a cross-composition of Set Splitting into itself.
- We may assume that the universes are of the same size, hence we think of them as of one, common universe.
- Assume that t is a power of 2 (by copying the instances).

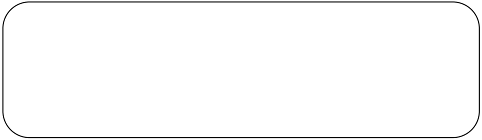
Cross-composing into SET SPLITTING

Input: Instances $(U, \mathcal{F}^i)$

Output: Instance $(U^*, \mathcal{F}^*)$

Input: Instances $(U, \mathcal{F}^i)$

Output: Instance $(U^*, \mathcal{F}^*)$



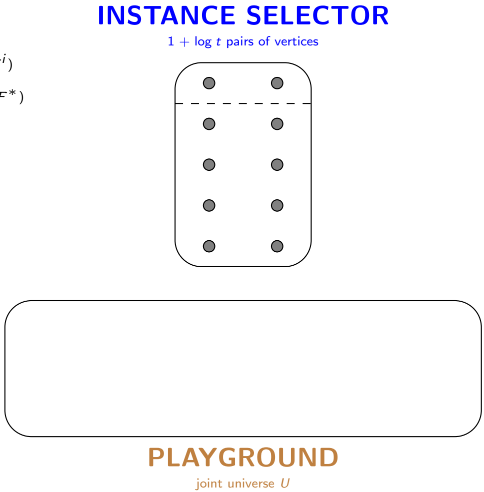
PLAYGROUND

joint universe U

Cross-composing into SET SPLITTING

Input: Instances $(U, \mathcal{F}^i)$

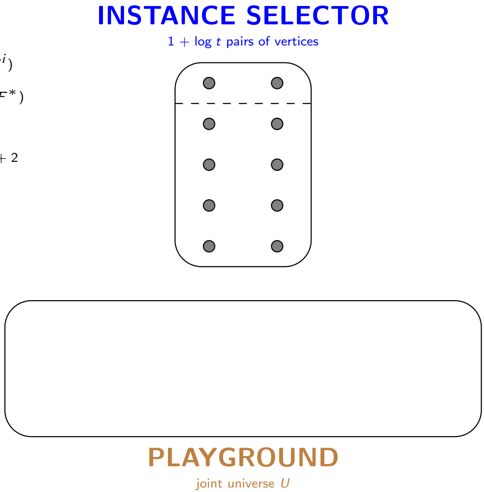
Output: Instance $(U^*, \mathcal{F}^*)$



Input: Instances $(U, \mathcal{F}^i)$

Output: Instance $(U^*, \mathcal{F}^*)$

$|U^*| = |U| + 2\log t + 2$

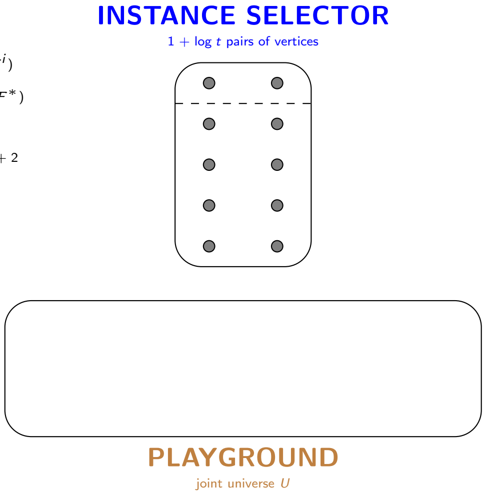


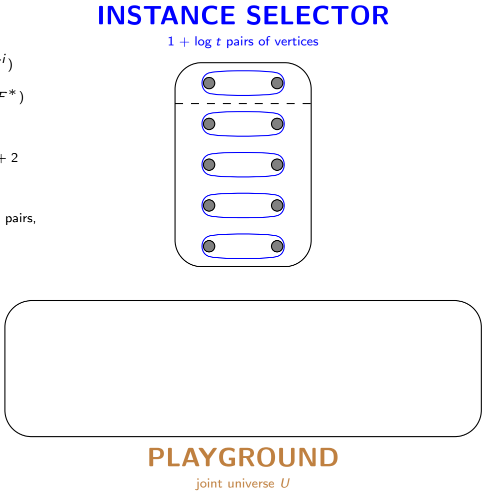
Input: Instances $(U, \mathcal{F}^i)$

Output: Instance $(U^*, \mathcal{F}^*)$

$|U^*| = |U| + 2\log t + 2$

$\mathcal{F}^*$ **consists of**:





Input: Instances $(U, \mathcal{F}^i)$

Output: Instance $(U^*, \mathcal{F}^*)$

$|U^*| = |U| + 2\log t + 2$

$\mathcal{F}^*$ **consists of**:
$1 + \log t$ 2-element sets for pairs,

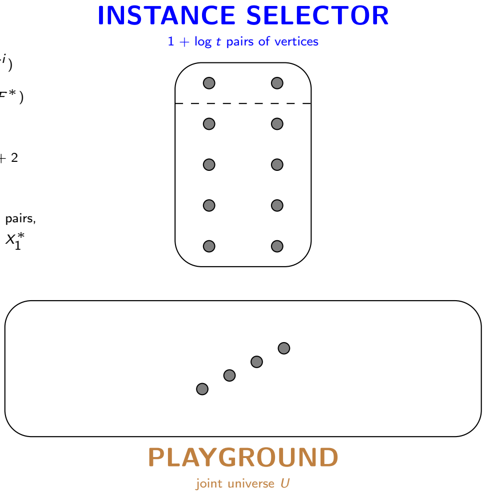
Cross-composing into Set Splitting

Input: Instances $(U, \mathcal{F}^i)$

Output: Instance $(U^*, \mathcal{F}^*)$

$|U^*| = |U| + 2\log t + 2$

$\mathcal{F}^*$ **consists of**:
$1 + \log t$ 2-element sets for pairs,
$\forall X \in \mathcal{F}^i$, two sets X_0^*, X_1^*

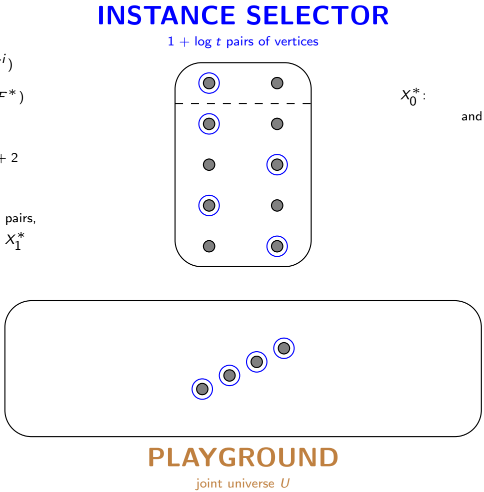


Input: Instances $(U, \mathcal{F}^i)$

Output: Instance $(U^*, \mathcal{F}^*)$

$|U^*| = |U| + 2\log t + 2$

$\mathcal{F}^*$ **consists of**:
$1 + \log t$ 2-element sets for pairs,
$\forall X \in \mathcal{F}^i$, two sets X_0^*, X_1^*



X_0^*: X, left special vertex, and binary encoding of i in IS

Cross-composing into SET SPLITTING

INSTANCE SELECTOR

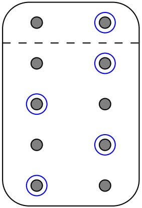
$1 + \log t$ pairs of vertices

Input: Instances $(U, \mathcal{F}^i)$

Output: Instance $(U^*, \mathcal{F}^*)$

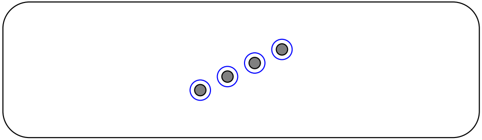
$|U^*| = |U| + 2\log t + 2$

$\mathcal{F}^*$ **consists of**:
$1 + \log t$ 2-element sets for pairs,
$\forall X \in \mathcal{F}^i$, two sets X_0^*, X_1^*



X_0^*: X, left special vertex, and binary encoding of i in IS

X_1^*: reverse X_0^* on IS



PLAYGROUND

joint universe U

Input: Instances $(U, \mathcal{F}^i)$

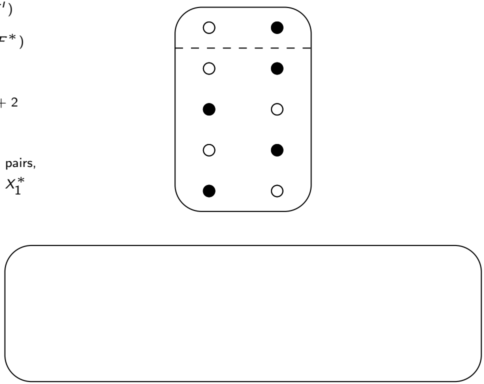
Output: Instance $(U^*, \mathcal{F}^*)$

$|U^*| = |U| + 2\log t + 2$

$\mathcal{F}^*$ **consists of**:
$1 + \log t$ 2-element sets for pairs,
$\forall X \in \mathcal{F}^i$, two sets X_0^*, X_1^*

INSTANCE SELECTOR

$1 + \log t$ pairs of vertices



Take any solution $\mathcal{C}$

PLAYGROUND

joint universe U

Input: Instances $(U, \mathcal{F}^i)$

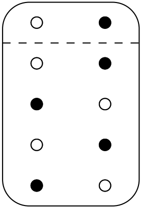
Output: Instance $(U^*, \mathcal{F}^*)$

$|U^*| = |U| + 2\log t + 2$

$\mathcal{F}^*$ **consists of**:
$1 + \log t$ 2-element sets for pairs,
$\forall X \in \mathcal{F}^i$, two sets X_0^*, X_1^*

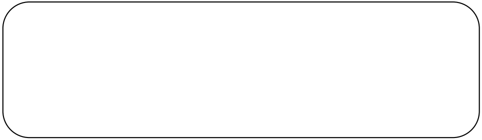
INSTANCE SELECTOR

$1 + \log t$ pairs of vertices



Take any solution $\mathcal{C}$

There is exactly one index i with monochromatic parts from IS.



PLAYGROUND

joint universe U

Cross-composing into SET SPLITTING

Input: Instances $(U, \mathcal{F}^i)$

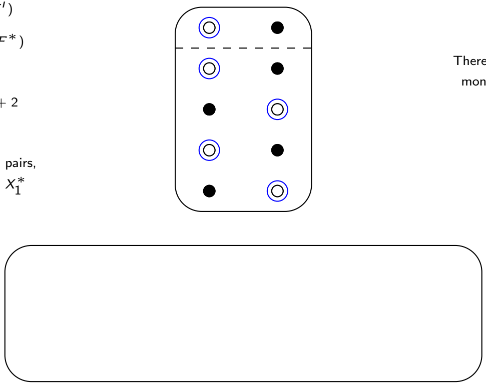
Output: Instance $(U^*, \mathcal{F}^*)$

$|U^*| = |U| + 2\log t + 2$

$\mathcal{F}^*$ **consists of**:
$1 + \log t$ 2-element sets for pairs,
$\forall X \in \mathcal{F}^i$, two sets X_0^*, X_1^*

INSTANCE SELECTOR

$1 + \log t$ pairs of vertices



PLAYGROUND

joint universe U

Take any solution $\mathcal{C}$

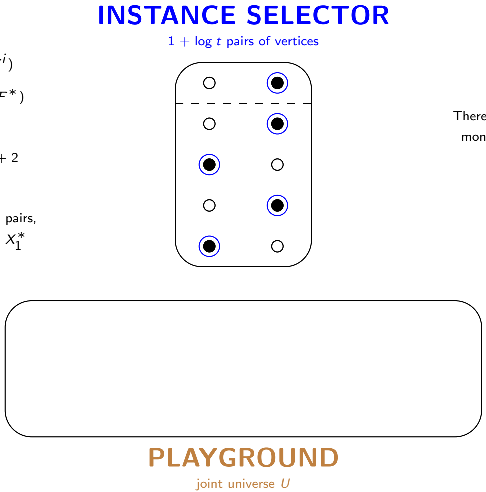
There is exactly one index i with monochromatic parts from IS.

Input: Instances $(U, \mathcal{F}^i)$

Output: Instance $(U^*, \mathcal{F}^*)$

$|U^*| = |U| + 2\log t + 2$

$\mathcal{F}^*$ **consists of**:
$1 + \log t$ 2-element sets for pairs,
$\forall X \in \mathcal{F}^i$, two sets X_0^*, X_1^*



Take any solution $\mathcal{C}$

There is exactly one index i with monochromatic parts from IS.

Input: Instances $(U, \mathcal{F}^i)$

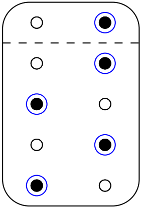
Output: Instance $(U^*, \mathcal{F}^*)$

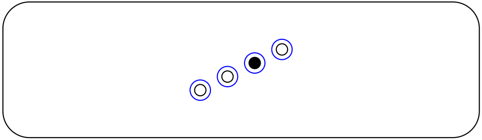
$|U^*| = |U| + 2\log t + 2$

$\mathcal{F}^*$ **consists of**:
$1 + \log t$ 2-element sets for pairs,
$\forall X \in \mathcal{F}^i$, two sets X_0^*, X_1^*

INSTANCE SELECTOR

$1 + \log t$ pairs of vertices





PLAYGROUND

joint universe U

Take any solution $\mathcal{C}$

There is exactly one index i with monochromatic parts from IS.

$(\Rightarrow)$: $\mathcal{C}$ on IS defines, which instance must be solved in PL

Cross-composing into SET SPLITTING

Input: Instances $(U, \mathcal{F}^i)$

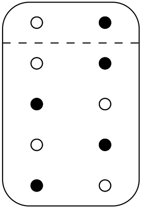
Output: Instance $(U^*, \mathcal{F}^*)$

$|U^*| = |U| + 2\log t + 2$

$\mathcal{F}^*$ **consists of**:
$1 + \log t$ 2-element sets for pairs,
$\forall X \in \mathcal{F}^i$, two sets X_0^*, X_1^*

INSTANCE SELECTOR

$1 + \log t$ pairs of vertices



PLAYGROUND

joint universe U

Take any solution $\mathcal{C}$

There is exactly one index i with monochromatic parts from IS.

($\Rightarrow$): $\mathcal{C}$ on IS defines, which instance must be solved in PL

($\Leftarrow$): If $(U, \mathcal{F}^i)$ is solvable, we set IS accordingly, and solve this instance in PL. Remaining sets are split for free.

SET SPLITTING: wrap up

- Unparameterized SET SPLITTING cross-composes into SET SPLITTING parameterized by $|U|$.

SET SPLITTING: wrap up

- Unparameterized SET SPLITTING cross-composes into SET SPLITTING parameterized by $|U|$.
- Unparameterized SET SPLITTING is **NP**-hard.

Set Splitting: wrap up

- Unparameterized Set Splitting cross-composes into Set Splitting parameterized by $|U|$.
- Unparameterized Set Splitting is **NP**-hard.
- Hence, Set Splitting parameterized by $|U|$ does not admit a polynomial kernel, unless $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.

SET SPLITTING: wrap up

- Unparameterized SET SPLITTING cross-composes into SET SPLITTING parameterized by $|U|$.
- Unparameterized SET SPLITTING is **NP**-hard.
- Hence, SET SPLITTING parameterized by $|U|$ does not admit a polynomial kernel, unless $\mathbf{NP} \subseteq \mathbf{coNP}/\text{poly}$.
- **Main lesson**:

SET SPLITTING: wrap up

- Unparameterized SET SPLITTING cross-composes into SET SPLITTING parameterized by $|U|$.
- Unparameterized SET SPLITTING is **NP**-hard.
- Hence, SET SPLITTING parameterized by $|U|$ does not admit a polynomial kernel, unless **NP** $\subseteq$ **coNP**/poly.
- **Main lesson**:
  - Model the **choice** of the instance to be solved.

SET SPLITTING: wrap up

- Unparameterized SET SPLITTING cross-composes into SET SPLITTING parameterized by $|U|$.
- Unparameterized SET SPLITTING is **NP**-hard.
- Hence, SET SPLITTING parameterized by $|U|$ does not admit a polynomial kernel, unless **NP** $\subseteq$ **coNP**/poly.
- **Main lesson**:
  - Model the **choice** of the instance to be solved.
  - **Idea**: choose $\log t$ bits of its index on an appropriate gadget.

SET SPLITTING: wrap up

- Unparameterized SET SPLITTING cross-composes into SET SPLITTING parameterized by $|U|$.
- Unparameterized SET SPLITTING is **NP**-hard.
- Hence, SET SPLITTING parameterized by $|U|$ does not admit a polynomial kernel, unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.
- **Main lesson**:
  - Model the **choice** of the instance to be solved.
  - **Idea**: choose $\log t$ bits of its index on an appropriate gadget.
  - Choice of the index makes the instance active, while the other instances are "switched off".

PPTs

- **Idea**: Hardness of kernelization can be transferred via reductions, similarly to **NP**-hardness.

PPTs

- **Idea**: Hardness of kernelization can be transferred via reductions, similarly to **NP**-hardness.

Polynomial parameter transformation (PPT)

A **polynomial parameter transformation** from a parameterized problem P to a parameterized problem Q is a polynomial-time algorithm that transforms a given instance (x, k) of P into an equivalent instance (x', k') of Q such that $k' = \mathrm{poly}(k)$.

PPTs

- **Idea**: Hardness of kernelization can be transferred via reductions, similarly to **NP**-hardness.

Polynomial parameter transformation (PPT)

A **polynomial parameter transformation** from a parameterized problem P to a parameterized problem Q is a polynomial-time algorithm that transforms a given instance (x, k) of P into an equivalent instance (x', k') of Q such that $k' = \mathrm{poly}(k)$.

Observation

If problem P PPT-reduces to Q, and P does not admit a polynomial compression algorithm (into any language R), then neither does Q.

PPTs

- **Idea**: Hardness of kernelization can be transferred via reductions, similarly to **NP**-hardness.

Polynomial parameter transformation (PPT)

A **polynomial parameter transformation** from a parameterized problem P to a parameterized problem Q is a polynomial-time algorithm that transforms a given instance (x, k) of P into an equivalent instance (x', k') of Q such that $k' = \mathrm{poly}(k)$.

Observation

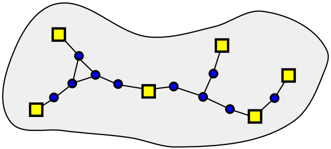
If problem P PPT-reduces to Q, and P does not admit a polynomial compression algorithm (into any language R), then neither does Q.

- **Proof**:
  Compose the PPT with the assumed compression for Q.

Application 2: Steiner Tree

Steiner Tree

I: Graph G with terminals $T \subseteq V(G)$, $k \in \mathbb{N}$
P: $k + |T|$
Q: Is there a set $X \subseteq V(G) \setminus T$, such that $|X| \leqslant k$ and $G[T \cup X]$ is connected?



Application 2: Steiner Tree

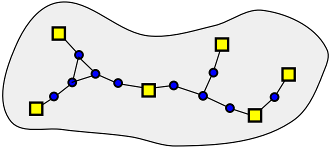
Steiner Tree

I: Graph G with terminals $T \subseteq V(G)$, $k \in \mathbb{N}$

P: $k + |T|$

Q: Is there a set $X \subseteq V(G) \setminus T$, such that $|X| \leqslant k$ and $G[T \cup X]$ is connected?

- We show that Steiner Tree has no polykernel (unless...) using a PPT from a auxiliary problem.



The auxiliary problem technique

- Introduce a simpler problem P, which is almost trivially compositional.

The auxiliary problem technique

- Introduce a simpler problem P, which is almost trivially compositional.
- Then design a PPT from P to the target problem.

The auxiliary problem technique

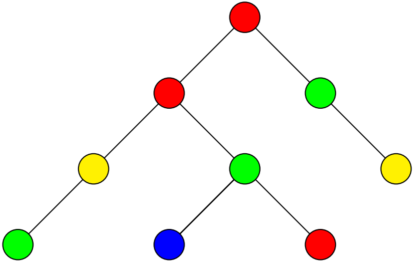
- Introduce a simpler problem P, which is almost trivially compositional.
- Then design a PPT from P to the target problem.
- **Idea**: Move the weight of the proof to the transformation and the actual definition of P.

The auxiliary problem technique

- Introduce a simpler problem P, which is almost trivially compositional.
- Then design a PPT from P to the target problem.
- **Idea**: Move the weight of the proof to the transformation and the actual definition of P.
- **High level**: Extract the essence of the original problem into the auxiliary problem.

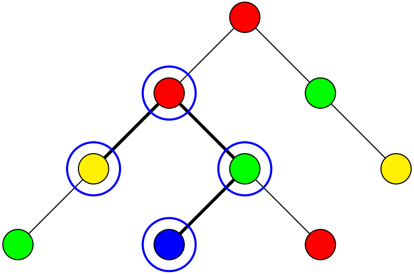
Colorful Graph Motif

I: Graph G and a coloring function $\phi \colon V(G) \to \{1, 2, \ldots, k\}$
P: k
Q: Does there exists a connected subgraph of G that contains exactly one vertex of each color?



Colorful Graph Motif

I: Graph G and a coloring function $\phi\colon V(G) \to \{1, 2, \ldots, k\}$
P: k
Q: Does there exists a connected subgraph of G that contains exactly one vertex of each color?



About CGM

- The problem is **NP**-hard even on trees.

About CGM

- The problem is **NP**-hard even on trees.
- FPT algorithms for various variants using the algebraic approach.

About CGM

- The problem is **NP**-hard even on trees.
- FPT algorithms for various variants using the algebraic approach.
- **Composition**: Take the disjoint union of instances, reuse colors.

About CGM

- The problem is **NP**-hard even on trees.
- FPT algorithms for various variants using the algebraic approach.
- **Composition**: Take the disjoint union of instances, reuse colors.
  - There is a connected colorful motif in the composed instance iff there is one in any of the input instances.

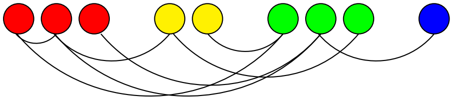
About CGM

- The problem is **NP**-hard even on trees.
- FPT algorithms for various variants using the algebraic approach.
- **Composition**: Take the disjoint union of instances, reuse colors.
  - There is a connected colorful motif in the composed instance iff there is one in any of the input instances.
- **Corollary**: no polykernel for CGM unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

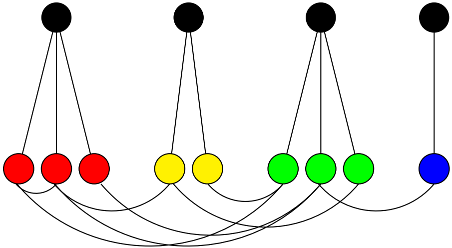
About CGM

- The problem is **NP**-hard even on trees.
- FPT algorithms for various variants using the algebraic approach.
- **Composition**: Take the disjoint union of instances, reuse colors.
  - There is a connected colorful motif in the composed instance iff there is one in any of the input instances.
- **Corollary**: no polykernel for CGM unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.
- **Now**: PPT from CGM to ST.

From CGM to ST



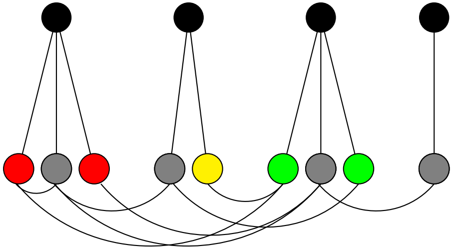
From CGM to ST



Attach a terminal to every color class.

Give budget k for connecting nodes.

From CGM to ST



Attach a terminal to every color class.

Give budget k for connecting nodes.

CGM: wrap up

- CGM has no polynomial kernel, unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

CGM: wrap up

- CGM has no polynomial kernel, unless **NP** $\subseteq$ **coNP**/poly.
- CGM PPT-reduces to Steiner Tree par. by $k + |T|$.

CGM: wrap up

- CGM has no polynomial kernel, unless **NP** $\subseteq$ **coNP**/poly.
- CGM PPT-reduces to Steiner Tree par. by $k + |T|$.
- Hence Steiner Tree par. by $k + |T|$ does not admit a polynomial kernel, unless **NP** $\subseteq$ **coNP**/poly.

CGM: wrap up

- CGM has no polynomial kernel, unless **NP** $\subseteq$ **coNP**/poly.
- CGM PPT-reduces to Steiner Tree par. by $k + |T|$.
- Hence Steiner Tree par. by $k + |T|$ does not admit a polynomial kernel, unless **NP** $\subseteq$ **coNP**/poly.
- **Note**: Composition for CGM is far simpler than trying to do this directly for Steiner Tree.

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.
- What about replacing it with, say, AND?

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.
- What about replacing it with, say, AND?
- **AND-distillation**, **AND-(cross)-composition**:
  Same as before, but with AND instead of OR.

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.
- What about replacing it with, say, AND?
- **AND-distillation**, **AND-(cross)-composition**:
  Same as before, but with AND instead of OR.
  - Example of problem admitting an AND-composition: TREEWIDTH.

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.
- What about replacing it with, say, AND?
- **AND-distillation**, **AND-(cross)-composition**:
  Same as before, but with AND instead of OR.
  - Example of problem admitting an AND-composition: TREEWIDTH.
- **AND-conjecture**:
  If $3\mathrm{SAT}$ has an AND-distillation, then $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.
- What about replacing it with, say, AND?
- **AND-distillation**, **AND-(cross)-composition**:
  Same as before, but with AND instead of OR.
  - Example of problem admitting an AND-composition: TREEWIDTH.
- **AND-conjecture**:
  If 3SAT has an AND-distillation, then **NP** $\subseteq$ **coNP**/poly.
  - The proof of Fortnow and Santhanam fails for AND.

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.
- What about replacing it with, say, AND?
- **AND-distillation**, **AND-(cross)-composition**:
  Same as before, but with AND instead of OR.
  - Example of problem admitting an AND-composition: TREEWIDTH.
- **AND-conjecture**:
  If $\mathrm{3SAT}$ has an AND-distillation, then **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.
  - The proof of Fortnow and Santhanam fails for AND.
  - The conjecture was proved by Drucker in 2012.

AND-compositions

- In the compositionality framework, we used the OR function to compose instances.
- What about replacing it with, say, AND?
- **AND-distillation**, **AND-(cross)-composition**:
  Same as before, but with AND instead of OR.
  - Example of problem admitting an AND-composition: TREEWIDTH.
- **AND-conjecture**:
  If $3\mathrm{SAT}$ has an AND-distillation, then **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.
  - The proof of Fortnow and Santhanam fails for AND.
  - The conjecture was proved by Drucker in 2012.
- **Corollary**: The whole framework works for AND instead of OR.

Weak compositions

- **Idea**: Inspect the proof of FS to get precise estimates.

Weak compositions

- **Idea**: Inspect the proof of FS to get precise estimates.
  - **Cor**: A framework for lower bounds on kernel sizes.

Weak compositions

- **Idea**: Inspect the proof of FS to get precise estimates.
  - **Cor**: A framework for lower bounds on kernel sizes.

Weak cross-composition

A **weak cross-composition of dimension** d from an unpar. problem Q to a par. problem L, is an algorithm that, given $\sim$-equivalent strings $x_1, x_2, \ldots, x_t$ for some polynomial equivalence relation $\sim$, in time $\mathrm{poly}\left(t + \sum_{i=1}^{t} |x_i|\right)$ produces one instance $(y, k^\star)$ such that

- $(y, k^\star) \in L$ iff $x_i \in Q$ for at least one $i = 1, 2, \ldots, t$,
- $k^\star = t^{1/d} \cdot \mathrm{poly}\left(\max_{i=1}^{t} |x_i|\right)$.

Weak compositions

- **Idea**: Inspect the proof of FS to get precise estimates.
  - **Cor**: A framework for lower bounds on kernel sizes.

Weak cross-composition

A **weak cross-composition of dimension** d from an unpar. problem Q to a par. problem L, is an algorithm that, given $\sim$-equivalent strings $x_1, x_2, \ldots, x_t$ for some polynomial equivalence relation $\sim$, in time $\mathrm{poly}\left(t + \sum_{i=1}^{t} |x_i|\right)$ produces one instance $(y, k^\star)$ such that

- $(y, k^\star) \in L$ iff $x_i \in Q$ for at least one $i = 1, 2, \ldots, t$,
- $k^\star = t^{1/d} \cdot \mathrm{poly}\left(\max_{i=1}^{t} |x_i|\right)$.

Weak cross-composition theorem

Suppose **NP** $\not\subseteq$ **coNP**/poly. If some **NP**-hard problem Q has a cross-composition of dimension d into L, then L does not admit a compression into any language R with bitsize $\mathcal{O}(k^{d-\varepsilon})$ for any $\varepsilon > 0$.

Weak compositions

- **Idea**: Inspect the proof of FS to get precise estimates.
  - **Cor**: A framework for lower bounds on kernel sizes.

Weak cross-composition

A **weak cross-composition of dimension** d from an unpar. problem Q to a par. problem L, is an algorithm that, given $\sim$-equivalent strings $x_1, x_2, \ldots, x_t$ for some polynomial equivalence relation $\sim$, in time $\mathrm{poly}\left(t + \sum_{i=1}^{t} |x_i|\right)$ produces one instance $(y, k^\star)$ such that

- $(y, k^\star) \in L$ iff $x_i \in Q$ for at least one $i = 1, 2, \ldots, t$,
- $k^\star = t^{1/d} \cdot \mathrm{poly}\left(\max_{i=1}^{t} |x_i|\right)$.

Weak cross-composition theorem

Suppose **NP** $\nsubseteq$ **coNP**$/\mathrm{poly}$. If some **NP**-hard problem Q has a cross-composition of dimension d into L, then L does not admit a compression into any language R with bitsize $\mathcal{O}(k^{d-\varepsilon})$ for any $\varepsilon > 0$.

- **Ex**: Vertex Cover has no compression into bitsize $\mathcal{O}(k^{2-\varepsilon})$.

Weak compositions

- **Idea**: Inspect the proof of FS to get precise estimates.
  - **Cor**: A framework for lower bounds on kernel sizes.

Weak cross-composition

A **weak cross-composition of dimension** d from an unpar. problem Q to a par. problem L, is an algorithm that, given $\sim$-equivalent strings $x_1, x_2, \ldots, x_t$ for some polynomial equivalence relation $\sim$, in time $\mathrm{poly}\left(t + \sum_{i=1}^{t} |x_i|\right)$ produces one instance $(y, k^\star)$ such that

- $(y, k^\star) \in L$ iff $x_i \in Q$ for at least one $i = 1, 2, \ldots, t$,
- $k^\star = t^{1/d} \cdot \mathrm{poly}\left(\max_{i=1}^{t} |x_i|\right)$.

Weak cross-composition theorem

Suppose **NP** $\not\subseteq$ **coNP**/poly. If some **NP**-hard problem Q has a cross-composition of dimension d into L, then L does not admit a compression into any language R with bitsize $\mathcal{O}(k^{d-\varepsilon})$ for any $\varepsilon > 0$.

- **Ex**: Vertex Cover has no compression into bitsize $\mathcal{O}(k^{2-\varepsilon})$.
- **Note**: The $2k$-kernel for VC needs $\mathcal{O}(k^2)$ bits for the encoding.

Conclusions

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.

Conclusions

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  - **Message 1**: What is hard for kernelization is **unbounded choice**.

Conclusions

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  - **Message 1**: What is hard for kernelization is **unbounded choice**.
  - **Message 2**: Turning intuition into a lower bound via cross-composition often needs a good understanding of the problem.

Conclusions

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  - **Message 1**: What is hard for kernelization is **unbounded choice**.
  - **Message 2**: Turning intuition into a lower bound via cross-composition often needs a good understanding of the problem.
- **Turing kernelization**: A Turing kernel is a poly-time algorithm that has oracle access to solving instances of size $\mathrm{poly}(k)$.

Conclusions

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  - **Message 1**: What is hard for kernelization is **unbounded choice**.
  - **Message 2**: Turning intuition into a lower bound via cross-composition often needs a good understanding of the problem.
- **Turing kernelization**: A Turing kernel is a poly-time algorithm that has oracle access to solving instances of size $\mathrm{poly}(k)$.
  - Composition framework does not apply.

Conclusions

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  - **Message 1**: What is hard for kernelization is **unbounded choice**.
  - **Message 2**: Turning intuition into a lower bound via cross-composition often needs a good understanding of the problem.
- **Turing kernelization**: A Turing kernel is a poly-time algorithm that has oracle access to solving instances of size $\mathrm{poly}(k)$.
  - Composition framework does not apply.
  - There are problems that have polynomial Turing kernels, but no polynomial kernel under **NP** $\not\subseteq$ **coNP**$/\mathrm{poly}$.

Conclusions

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  - **Message 1**: What is hard for kernelization is **unbounded choice**.
  - **Message 2**: Turning intuition into a lower bound via cross-composition often needs a good understanding of the problem.
- **Turing kernelization**: A Turing kernel is a poly-time algorithm that has oracle access to solving instances of size $\mathrm{poly}(k)$.
  - Composition framework does not apply.
  - There are problems that have polynomial Turing kernels, but no polynomial kernel under $\mathbf{NP} \not\subseteq \mathbf{coNP}/\mathrm{poly}$.
  - **Open**: A technique for ruling out Turing kernels.

Conclusions

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  - **Message 1**: What is hard for kernelization is **unbounded choice**.
  - **Message 2**: Turning intuition into a lower bound via cross-composition often needs a good understanding of the problem.
- **Turing kernelization**: A Turing kernel is a poly-time algorithm that has oracle access to solving instances of size $\mathrm{poly}(k)$.
  - Composition framework does not apply.
  - There are problems that have polynomial Turing kernels, but no polynomial kernel under $\mathbf{NP} \not\subseteq \mathbf{coNP}/\mathrm{poly}$.
  - **Open**: A technique for ruling out Turing kernels.
- **Complexity theory for kernelization**: Using PPT as reductions, one can build a hierarchy of complexity classes [HKSWW].

Conclusions

- **Composition**: a versatile framework for proving lower bounds for polynomial kernelization.
  - **Message 1**: What is hard for kernelization is **unbounded choice**.
  - **Message 2**: Turning intuition into a lower bound via cross-composition often needs a good understanding of the problem.
- **Turing kernelization**: A Turing kernel is a poly-time algorithm that has oracle access to solving instances of size $\mathrm{poly}(k)$.
  - Composition framework does not apply.
  - There are problems that have polynomial Turing kernels, but no polynomial kernel under $\mathbf{NP} \not\subseteq \mathbf{coNP}/\mathrm{poly}$.
  - **Open**: A technique for ruling out Turing kernels.
- **Complexity theory for kernelization**: Using PPT as reductions, one can build a hierarchy of complexity classes [HKSWW].
- **Thank you for your attention!**