Lower bounds for polynomial kernelization

Part 1

Michał Pilipczuk

Institutt for Informatikk, Universitetet i Bergen

August 19th, 2014

Outline

- **Goal**: how to prove that some problems do **not** admit polynomial kernelization algorithms?
- **Part 1:**
  - Introduction of the (cross)-composition framework.
  - Basic examples.
- **Part 2**:
  - PPT reductions.
  - Case study of several cross-compositions.
  - Weak compositions.

Disclaimer

- **This will be a complexity theory lecture.**

Disclaimer

- **This will be a complexity theory lecture.**
- Unparameterized problems = languages over Σ = subsets of $\Sigma^\star$, for a constant size alphabet Σ.

Disclaimer

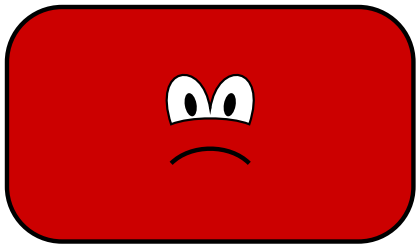
- **This will be a complexity theory lecture.**
- Unparameterized problems = languages over Σ = subsets of $\Sigma^\star$, for a constant size alphabet Σ.
- Parameterized problems are sets of pairs (x, k), where $x \in \Sigma^\star$ and k is a nonnegative integer.

Disclaimer

- **This will be a complexity theory lecture.**
- Unparameterized problems = languages over Σ = subsets of $\Sigma^\star$, for a constant size alphabet Σ.
- Parameterized problems are 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.

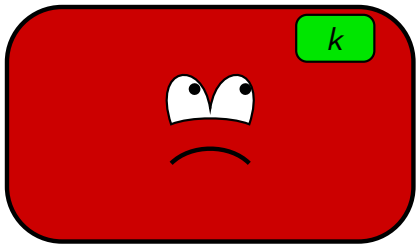
Kernelization — recap

Kernelization — recap



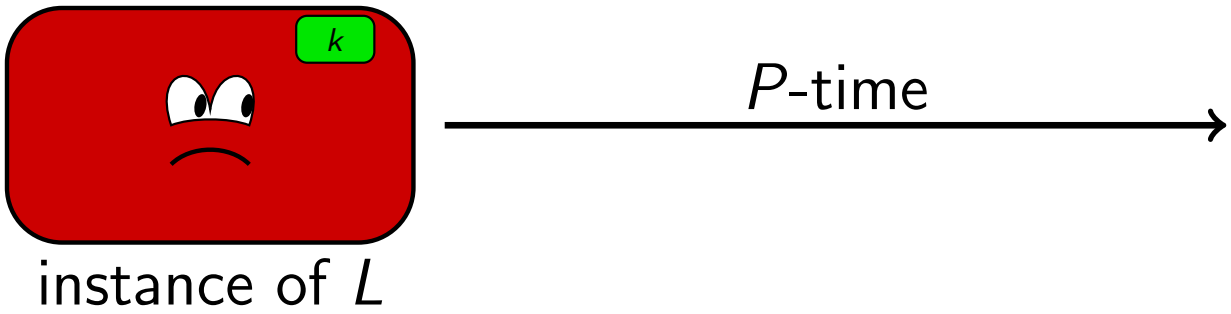
instance of L

Kernelization — recap

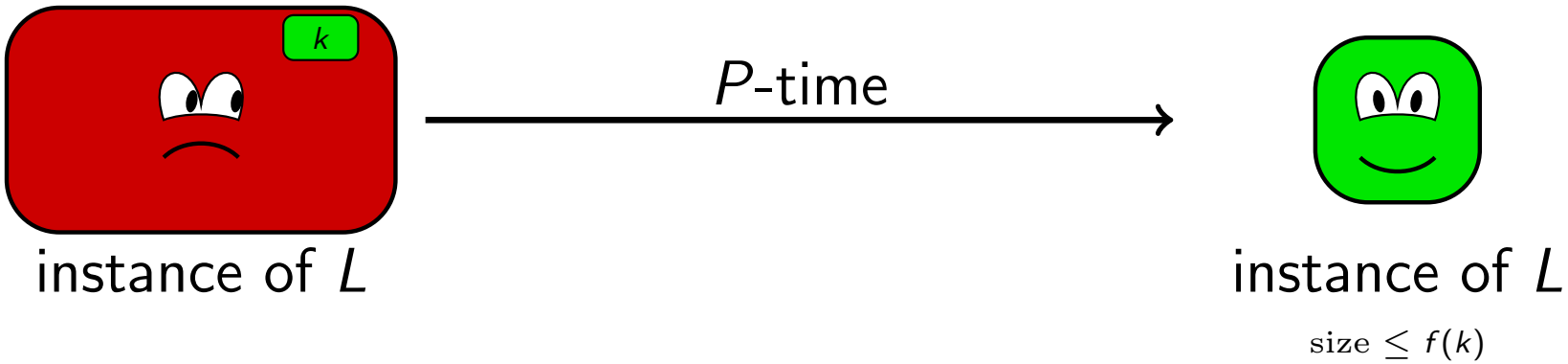


instance of L

Kernelization — recap



Kernelization — recap



Kernelization and FPT

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

Kernelization and FPT

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

Kernelization and FPT

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

Kernelization and FPT

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

Kernelization and FPT

- If a decidable problem admits a kernelization algorithm, then it is FPT.
- Any FPT problem admits a kernelization algorithm:
  - Let (x, k) be the input instance.
  - If $|x| \leq f(k)$, 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 admits a kernelization algorithm, then it is FPT.
- Any FPT problem admits a kernelization algorithm:
  - Let (x, k) be the input instance.
  - If $|x| \leq f(k)$, 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 admits a kernelization algorithm, then it is FPT.
- Any FPT problem admits a kernelization algorithm:
  - Let (x, k) be the input instance.
  - If $|x| \leq f(k)$, 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 kernel that has always, say, 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 kernel that has always, say, 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 kernel that has always, say, 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 kernel that has always, say, 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.

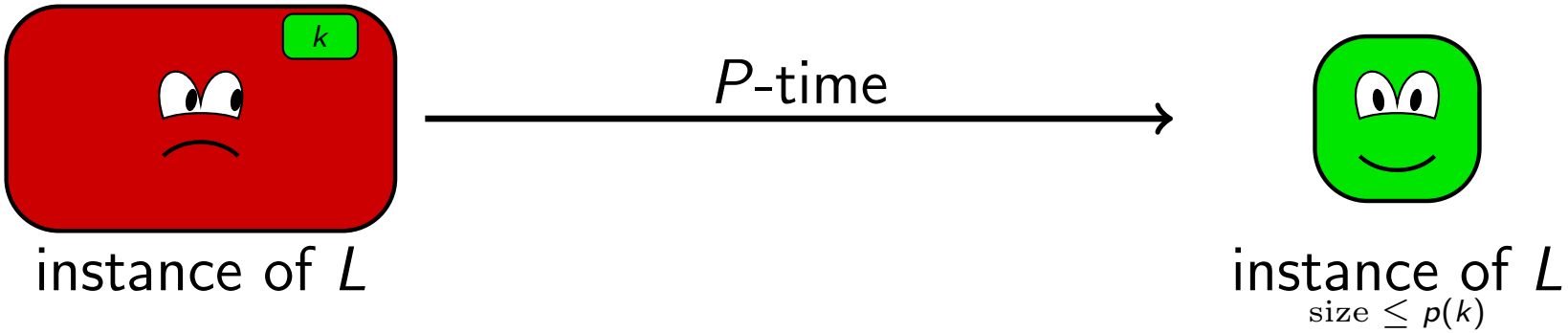
Motivating intuition

- **Intuition**: The final number of bits is much less than the number input instances. Most of the instances must have been **discarded completely**.

Kernelization and Compression

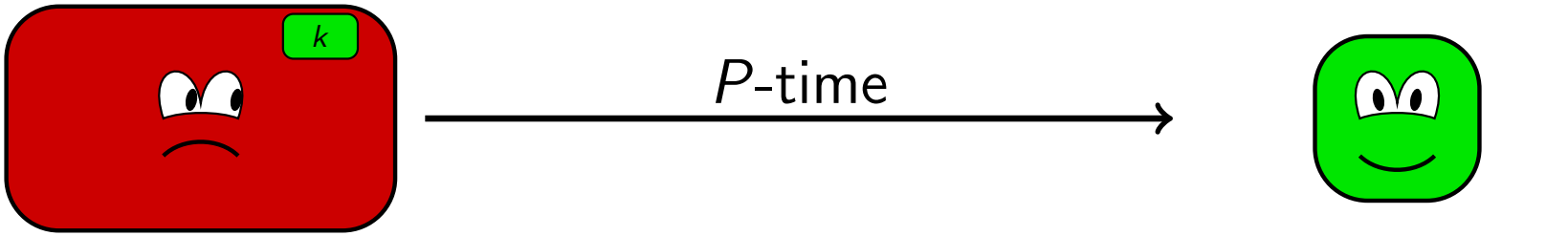
Kernelization and Compression

KERNELIZATION

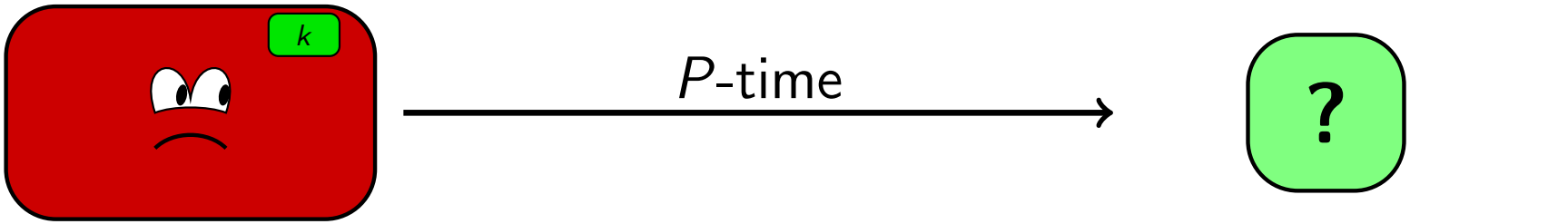


Kernelization and Compression

KERNELIZATION

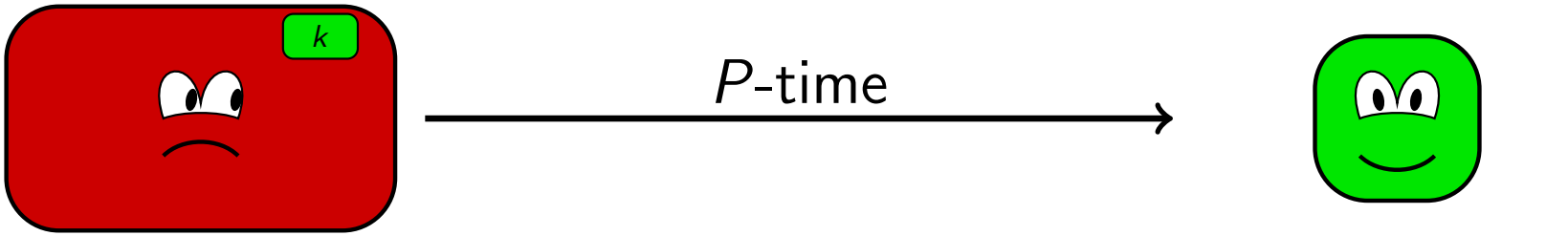


COMPRESSION

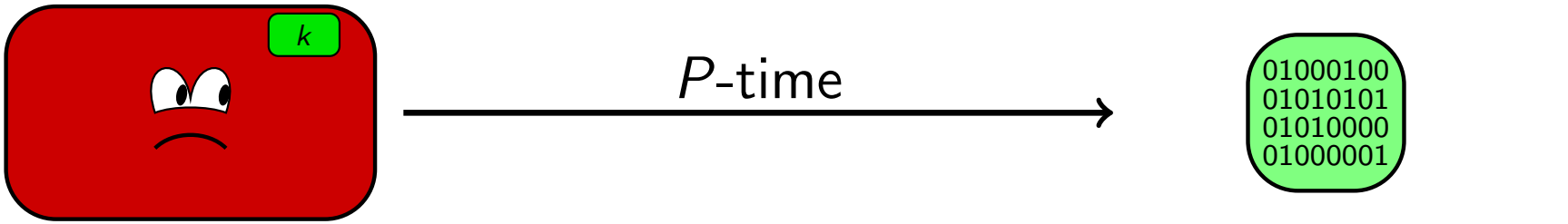


Kernelization and Compression

KERNELIZATION



COMPRESSION



Kernelization and Compression

- A polynomial kernelization is always a polynomial compression.

Kernelization and Compression

- 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

- 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

- 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

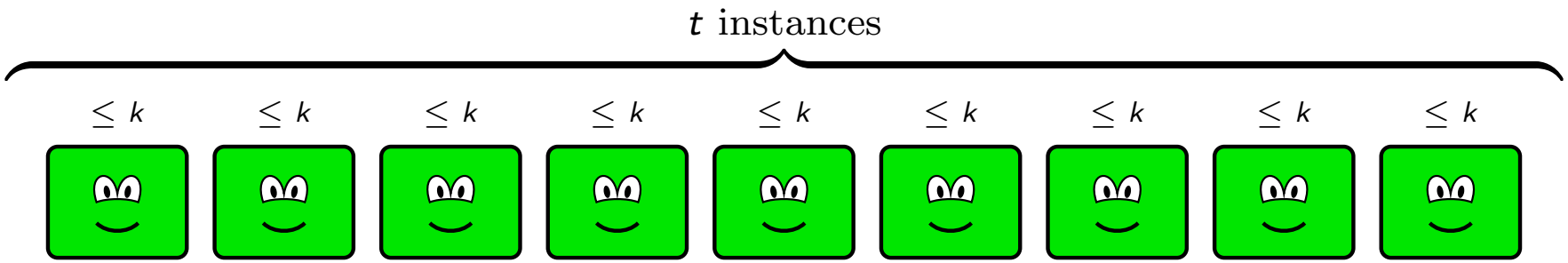
- Let L, R be *unparameterized* languages.

OR-distillation of L into R

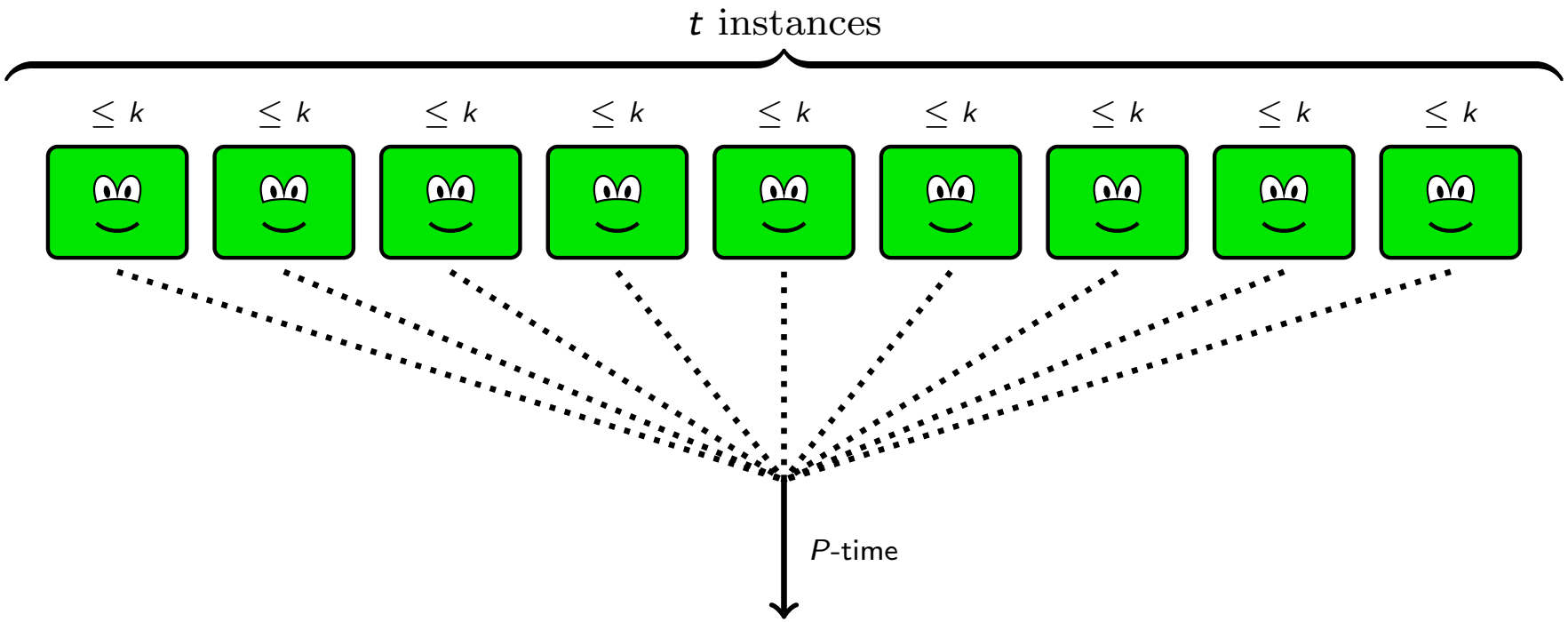
Input: Strings $x_1, x_2, \ldots, x_t$, each of length at most k.
Time: $\mathrm{poly}(t + \sum_{i=1}^{t} |x_i|)$.
Output: One string y such that
(a) $|y| = \mathrm{poly}(k)$, and
(b) $y \in R$ if and only if $x_i \in L$ for at least one i.

OR-distillation on picture

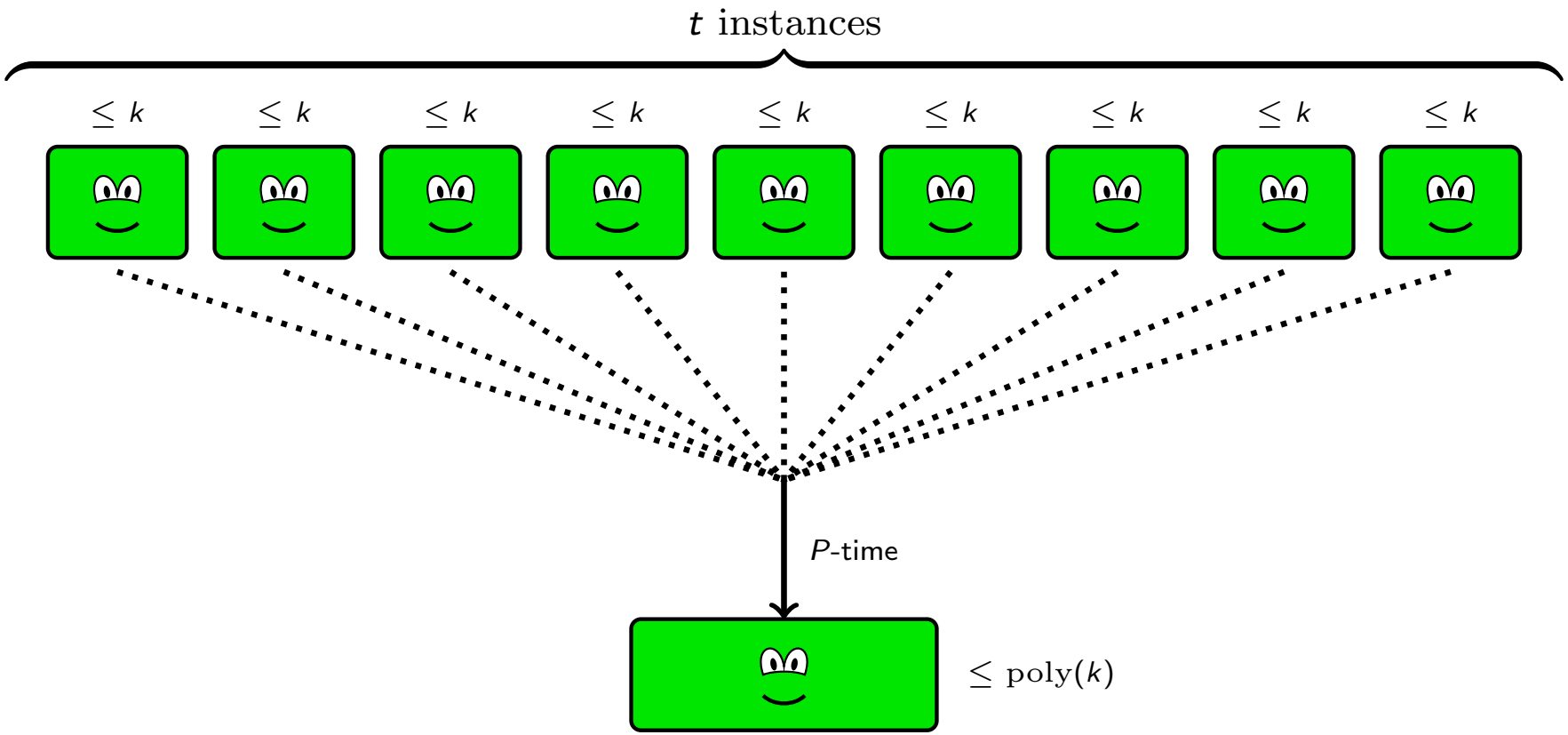
OR-distillation on picture



OR-distillation on picture



OR-distillation on picture



OR-distillation

- **Grocery intuition**:

OR-distillation

- **Grocery intuition**:
  - Suppose input instances are apples, and the OR-distillation algorithm is a blender.

OR-distillation

- **Grocery intuition**:
  - Suppose input instances are apples, and the OR-distillation algorithm is a blender.
  - If one of the apples was rotten, then the blend must be untasty.

OR-distillation

- **Grocery intuition**:
  - Suppose input instances are apples, and the OR-distillation algorithm is a blender.
  - If one of the apples was rotten, then the blend must be untasty.
  - If the blend is much smaller than the total input fruit mass, then it will be possible that a computationally too weak blender will lose the rotten apple.

OR-distillation

- **Grocery intuition**:
  - Suppose input instances are apples, and the OR-distillation algorithm is a blender.
  - If one of the apples was rotten, then the blend must be untasty.
  - If the blend is much smaller than the total input fruit mass, then it will be possible that a computationally too weak blender will lose the rotten apple.
- OR-L: language of strings $x_1\#x_2\#\ldots\#x_t$ such that $x_i \in L$ for at least one i.

OR-distillation

- **Grocery intuition**:
  - Suppose input instances are apples, and the OR-distillation algorithm is a blender.
  - If one of the apples was rotten, then the blend must be untasty.
  - If the blend is much smaller than the total input fruit mass, then it will be possible that a computationally too weak blender will lose the rotten apple.
- OR-L: language of strings $x_1\#x_2\#\ldots\#x_t$ such that $x_i \in L$ for at least one i.
- OR-distillation of L into R is a polynomial compression of OR-L into R, where OR-L is parameterized by $\max |x_i|$.

Backbone theorem

OR-distillation theorem Fortnow, Santhanam; STOC 2008, JCSS 2011

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; STOC 2008, JCSS 2011

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 $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.

The assumption

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

The assumption

- Assumption $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$ may seem mysterious.
- More known variant: $\mathbf{NP} \neq \mathbf{coNP}$.

The assumption

- Assumption $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$ may seem mysterious.
- More known variant: $\mathbf{NP} \neq \mathbf{coNP}$.
  - Verifying proofs in **P**-time is not equivalent to veryfying counterexamples in **P**-time.

The assumption

- Assumption $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$ may seem mysterious.
- More known variant: $\mathbf{NP} \neq \mathbf{coNP}$.
  - Verifying proofs in **P**-time is not equivalent to veryfying counterexamples in **P**-time.
- $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$ is strengthening of this by saying that verifying proofs cannot be simulated by verifying counterexamples even is we allow *polynomial advice*.

The assumption

- Assumption $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$ may seem mysterious.
- More known variant: $\mathbf{NP} \neq \mathbf{coNP}$.
  - Verifying proofs in **P**-time is not equivalent to veryfying counterexamples in **P**-time.
- $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$ is strengthening of this by saying that verifying proofs cannot be simulated by verifying counterexamples even is we allow *polynomial advice*.
- It is known that $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$ implies that $\mathrm{PH} = \Sigma_3^{\mathrm{P}}$.

The assumption

- Assumption $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$ may seem mysterious.
- More known variant: $\mathbf{NP} \neq \mathbf{coNP}$.
  - Verifying proofs in **P**-time is not equivalent to veryfying counterexamples in **P**-time.
- $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$ is strengthening of this by saying that verifying proofs cannot be simulated by verifying counterexamples even is we allow *polynomial advice*.
- It is known that $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$ implies that $\mathrm{PH} = \Sigma_3^{\mathrm{P}}$.
- Not as bad as $\mathbf{P} = \mathbf{NP}$, but pretty severe.

A glimpse into the proof

- The proof is purely information-theoretical.

A glimpse into the proof

- The proof is purely information-theoretical.
- Intuitively, the space for possible kernels is too small to store information about very long sequences of instances.

A glimpse into the proof

- The proof is purely information-theoretical.
- Intuitively, the space for possible kernels is too small to store information about very long sequences of instances.
- An algorithm in **P** cannot guess, which instance is more prone to have a positive answer, so we need to store information about **all** of them.

A glimpse into the proof

- The proof is purely information-theoretical.
- Intuitively, the space for possible kernels is too small to store information about very long sequences of instances.
- An algorithm in **P** cannot guess, which instance is more prone to have a positive answer, so we need to store information about **all** of them.
- **Main trick**:

A glimpse into the proof

- The proof is purely information-theoretical.
- Intuitively, the space for possible kernels is too small to store information about very long sequences of instances.
- An algorithm in **P** cannot guess, which instance is more prone to have a positive answer, so we need to store information about **all** of them.
- **Main trick**:
  - show that the space for kernels is so small that one can find a linear number of *representative* kernels;

A glimpse into the proof

- The proof is purely information-theoretical.
- Intuitively, the space for possible kernels is too small to store information about very long sequences of instances.
- An algorithm in **P** cannot guess, which instance is more prone to have a positive answer, so we need to store information about **all** of them.
- **Main trick**:
  - show that the space for kernels is so small that one can find a linear number of *representative* kernels;
  - plug these kernels as the advice to a **coNP**-algorithm for SAT.

A glimpse into the proof

- The proof is purely information-theoretical.
- Intuitively, the space for possible kernels is too small to store information about very long sequences of instances.
- An algorithm in **P** cannot guess, which instance is more prone to have a positive answer, so we need to store information about **all** of them.
- **Main trick**:
  - show that the space for kernels is so small that one can find a linear number of *representative* kernels;
  - plug these kernels as the advice to a **coNP**-algorithm for SAT.
- **Look into the book.**

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: $\mathrm{poly}(t + \sum_{i=1}^{t} |x_i| + k)$.

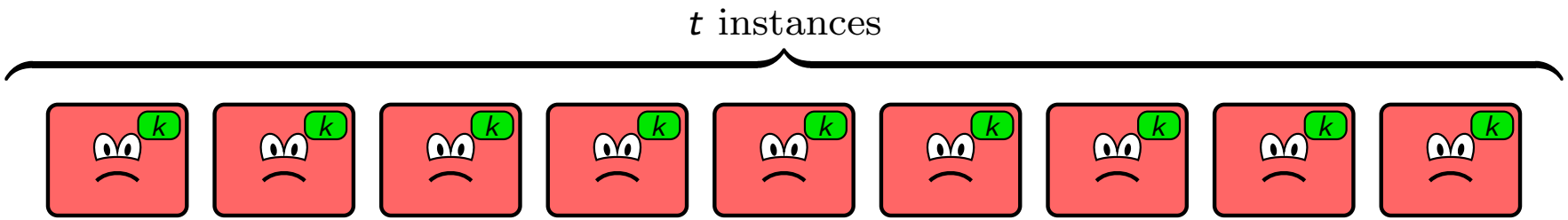
Output: One instance $(y, k^\star)$ such that

(a) $k^\star = \mathrm{poly}(k)$, and

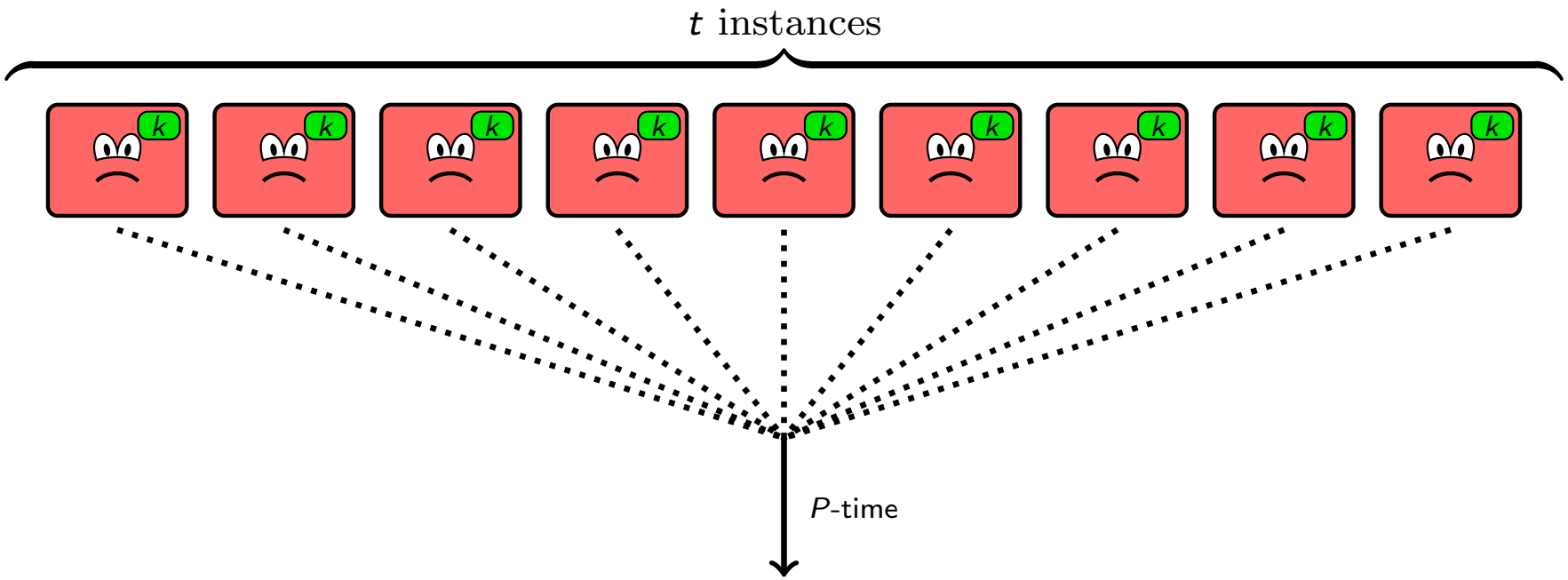
(b) $(y, k^\star) \in L$ iff $(x_i, k) \in L$ for at least one i.

OR-composition on picture

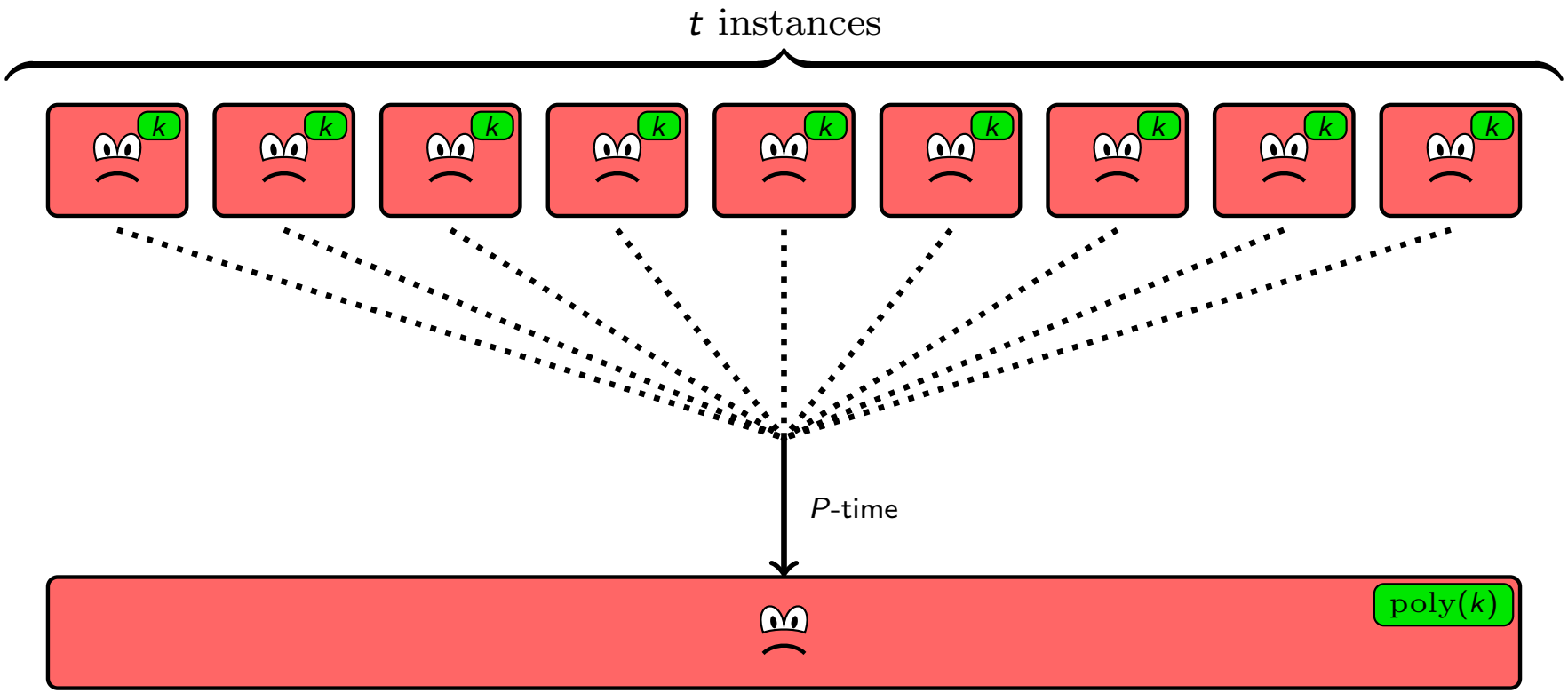
OR-composition on picture



OR-composition on picture



OR-composition on picture



OR-composition theorem

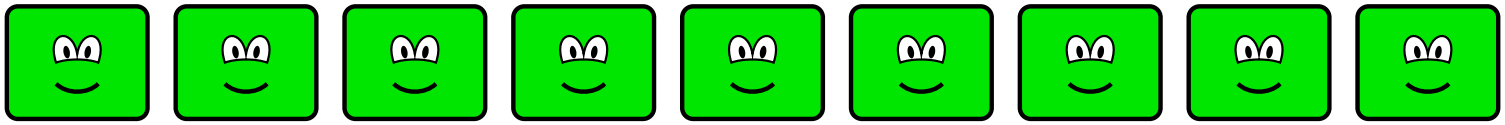
OR-composition theorem Bodlaender et al.; ICALP 2008, JCSS 2009

If 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 $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.

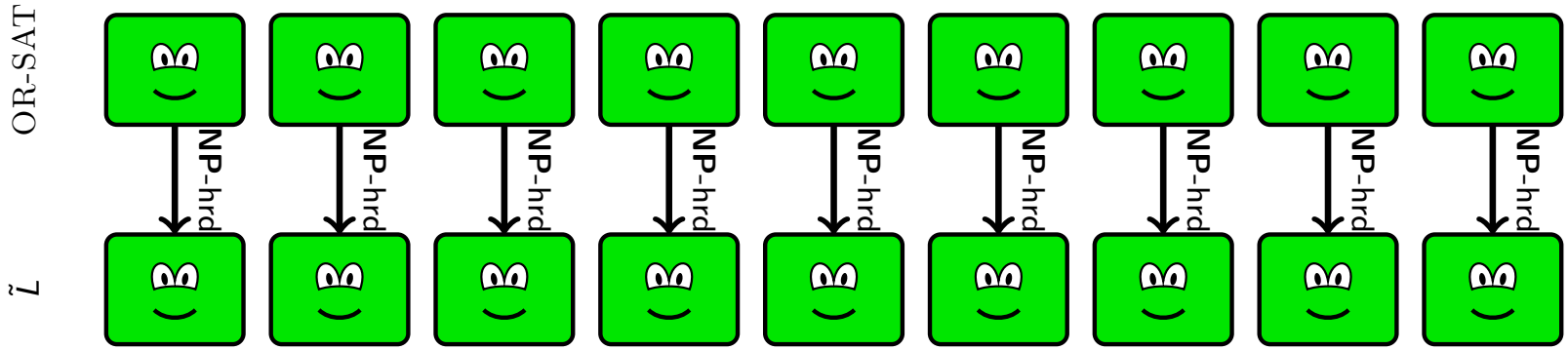
Proof

Proof

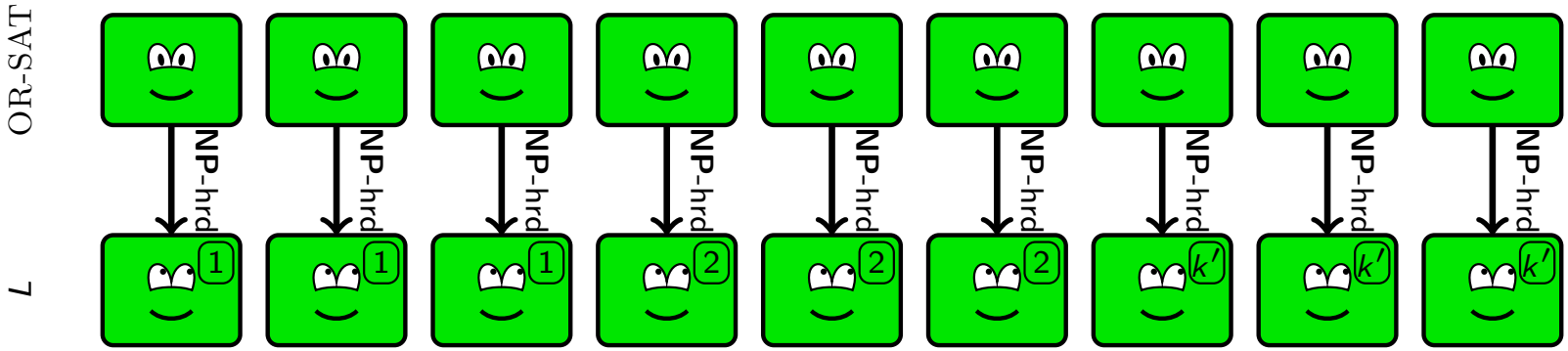
OR-SAT



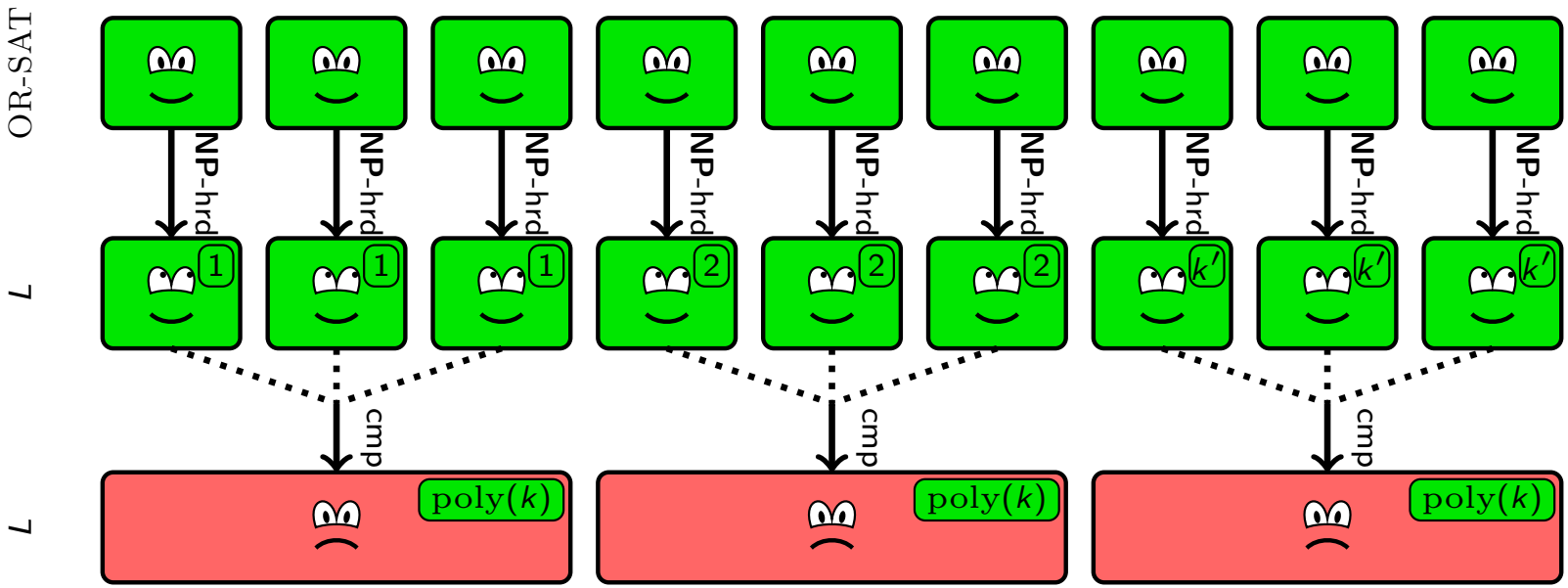
Proof


OR-SAT
$\tilde{L}$
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd

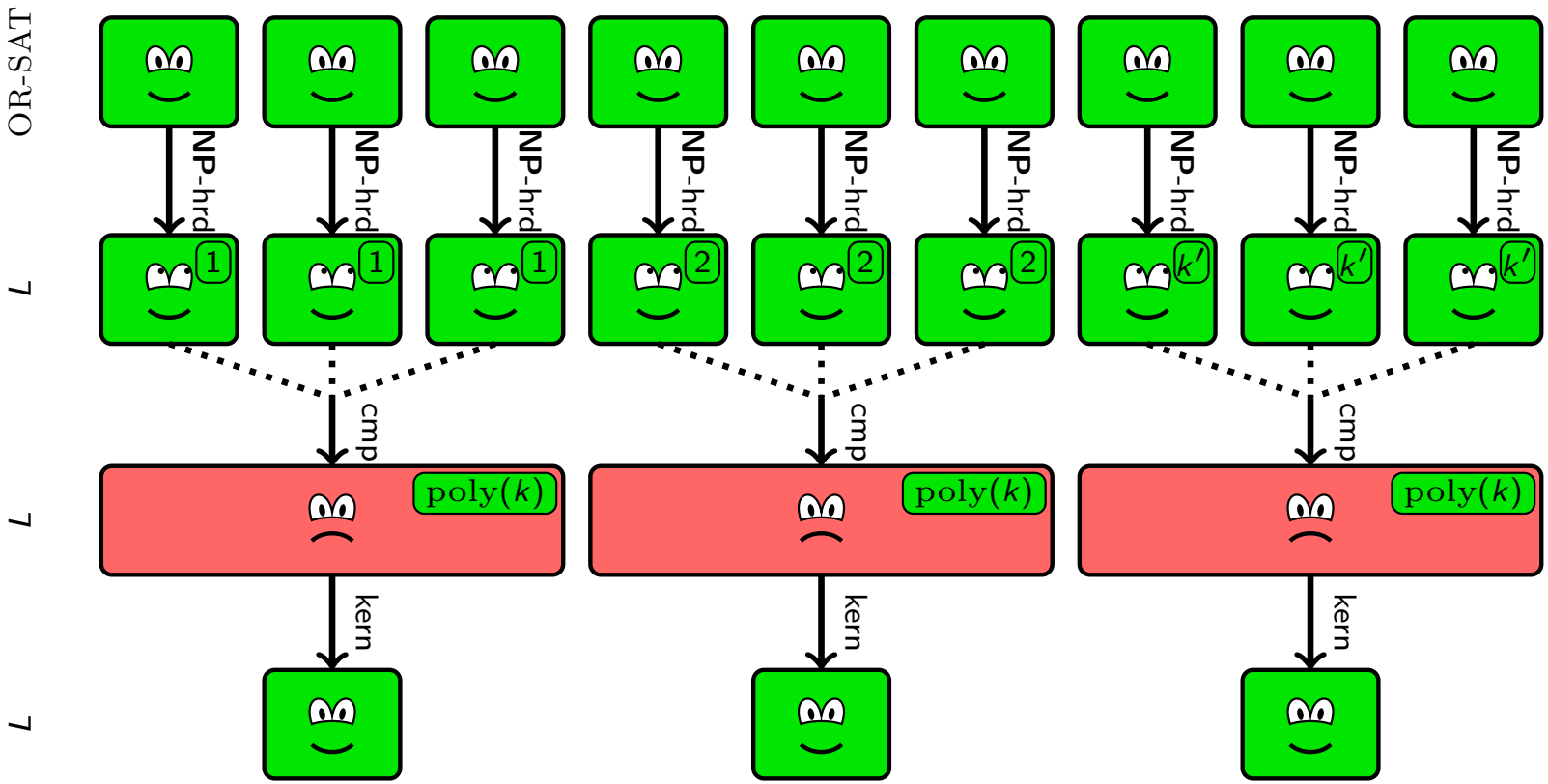
Proof



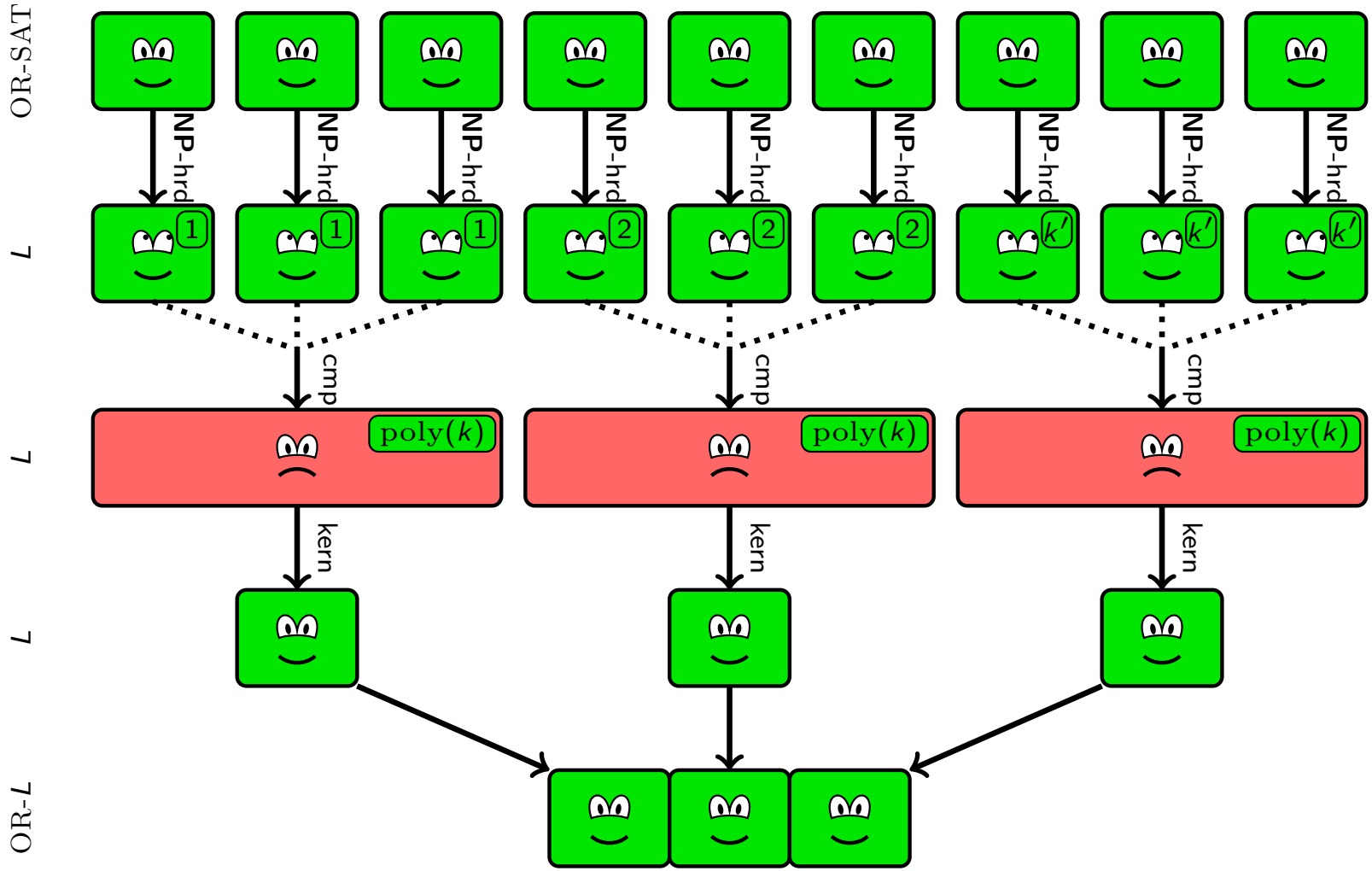
Proof


OR-SAT
L
L
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
1
1
1
2
2
2
k′
k′
k′
cmp
cmp
cmp
poly(k)
poly(k)
poly(k)

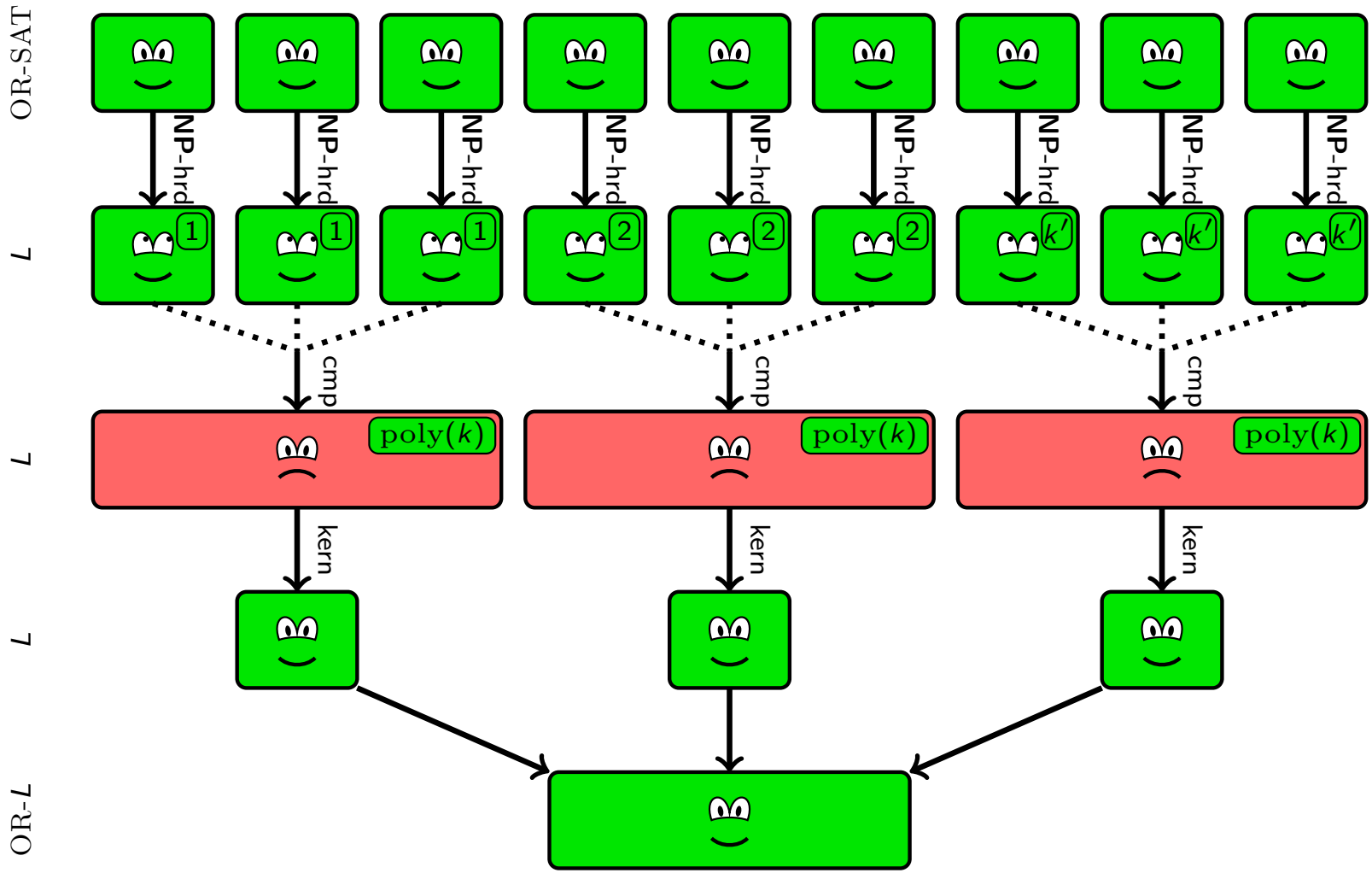
Proof


OR-SAT
L
L
L
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
1
1
1
2
2
2
k′
k′
k′
cmp
cmp
cmp
poly(k)
poly(k)
poly(k)
kern
kern
kern

Proof



Proof


OR-SAT
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
NP-hrd
L
1
1
1
2
2
2
k′
k′
k′
L
cmp
cmp
cmp
poly(k)
poly(k)
poly(k)
L
kern
kern
kern
OR-L

Corollaries

- k-Path does not admit poly-kernel, unless **NP** $\subseteq$ **coNP**/poly.

Corollaries

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

Corollaries

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

Corollaries

- k-PATH does not admit poly-kernel, unless **NP** $\subseteq$ **coNP**/poly.
- **Composition**: Take disjoint union of graphs and the same parameter.
  - A graph admits a k-path iff any of its connected components does.
- Same for k-CYCLE; this opens a bag of results.

Corollaries

- k-PATH does not admit poly-kernel, unless **NP** $\subseteq$ **coNP**/poly.
- **Composition**: Take disjoint union of graphs and the same parameter.
  - A graph admits a k-path iff any of its connected components does.
- Same for k-CYCLE; this opens a bag of results.
- Today, investigating the existence of a polynomial kernel is an immediate second goal after showing that a problem is FPT.

Adding features

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

Adding features

- Does the proof actually exclude even polynomial compression, 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, 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, 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 language Q into one instance of L.

Adding features

- Does the proof actually exclude even polynomial compression, 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 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, 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 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 more features

- How large can t be?

Adding more features

- How large can t be?
- Well, not larger than $(|\Sigma| + 1)^k$, as we may remove duplicates of the input instances.

Adding more features

- How large can t be?
- Well, not larger than $(|\Sigma| + 1)^k$, as we may remove duplicates of the input instances.
- Hence, we may assume that $\log t = \mathcal{O}(k)$,

Adding more features

- How large can t be?
- Well, not larger than $(|\Sigma| + 1)^k$, as we may remove duplicates of the input instances.
- Hence, we may assume that $\log t = \mathcal{O}(k)$,
- which means that the parameter of the composed instance may depend polynomially on **both** k and $\log t$.

Adding more features

- How large can t be?
- Well, not larger than $(|\Sigma| + 1)^k$, as we may remove duplicates of the input instances.
- Hence, we may assume that $\log t = \mathcal{O}(k)$,
- which means that 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; ICALP 2009).

After invention of the composition framework

- A huge amount of no-poly-kernel results.

After invention of the composition framework

- A huge amount of no-poly-kernel results.
- Most of the works use a subset of mentioned features.

After invention of the composition framework

- A huge amount of no-poly-kernel results.
- Most of the works use a subset of mentioned features.
- STACS 2011: Bodlaender, Jansen, and Kratsch propose a new formalism, dubbed **cross-composition**, that gathers all these features.

Polynomial equivalence relation

Polynomial equivalence relation

An equivalence relation $\mathcal{R}$ on $\Sigma^\star$ is called a polynomial equivalence relation if the following two conditions hold:

- Checking whether two strings $x, y \in \Sigma^\star$ are $\mathcal{R}$-equivalent can be done in $\mathrm{poly}(|x| + |y|)$ time.
- $\mathcal{R}$ partitions strings of length at most n into $\mathrm{poly}(n)$ equivalence classes.

Polynomial equivalence relation

Polynomial equivalence relation

An equivalence relation $\mathcal{R}$ on $\Sigma^\star$ is called a polynomial equivalence relation if the following two conditions hold:

- Checking whether two strings $x, y \in \Sigma^\star$ are $\mathcal{R}$-equivalent can be done in $\mathrm{poly}(|x| + |y|)$ time.
- $\mathcal{R}$ partitions strings of length at most n into $\mathrm{poly}(n)$ equivalence classes.

- **Examples**:

Polynomial equivalence relation

Polynomial equivalence relation

An equivalence relation $\mathcal{R}$ on $\Sigma^\star$ is called a polynomial equivalence relation if the following two conditions hold:

- Checking whether two strings $x, y \in \Sigma^\star$ are $\mathcal{R}$-equivalent can be done in $\mathrm{poly}(|x| + |y|)$ time.
- $\mathcal{R}$ partitions strings of length at most n into $\mathrm{poly}(n)$ equivalence classes.

- **Examples**:
  - partitioning with respect to the number of vertices of the graph;

Polynomial equivalence relation

Polynomial equivalence relation

An equivalence relation $\mathcal{R}$ on $\Sigma^\star$ is called a polynomial equivalence relation if the following two conditions hold:

- Checking whether two strings $x, y \in \Sigma^\star$ are $\mathcal{R}$-equivalent can be done in $\mathrm{poly}(|x| + |y|)$ time.
- $\mathcal{R}$ partitions strings of length at most n into $\mathrm{poly}(n)$ equivalence classes.

- **Examples**:
  - 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 $\mathcal{R}$ and an algorithm that, given $\mathcal{R}$-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 $\mathcal{R}$ and an algorithm that, given $\mathcal{R}$-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

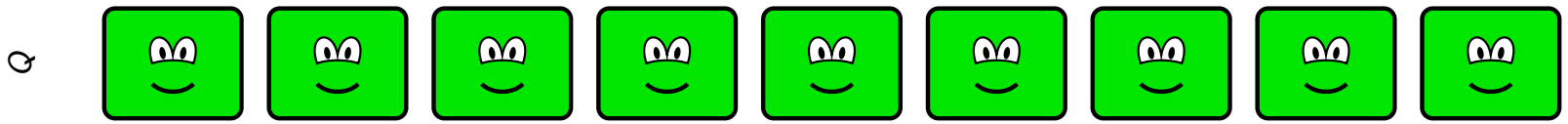
Bodlaender et al.; STACS 2011, SIDMA 2014

If some **NP**-hard problem Q cross-composes into L, then L does not admit a polynomial compression into any language R, unless $\mathbf{NP} \subseteq \mathbf{coNP}/\mathrm{poly}$.

Proof

Proof

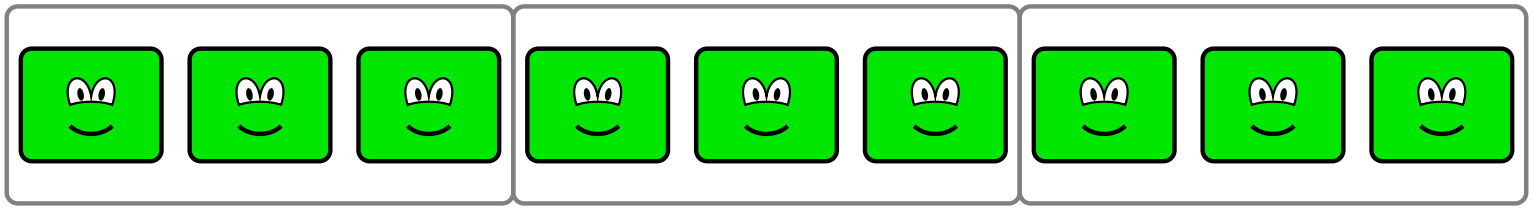
$k = \max |x_i|, \quad \log t = \mathcal{O}(k)$



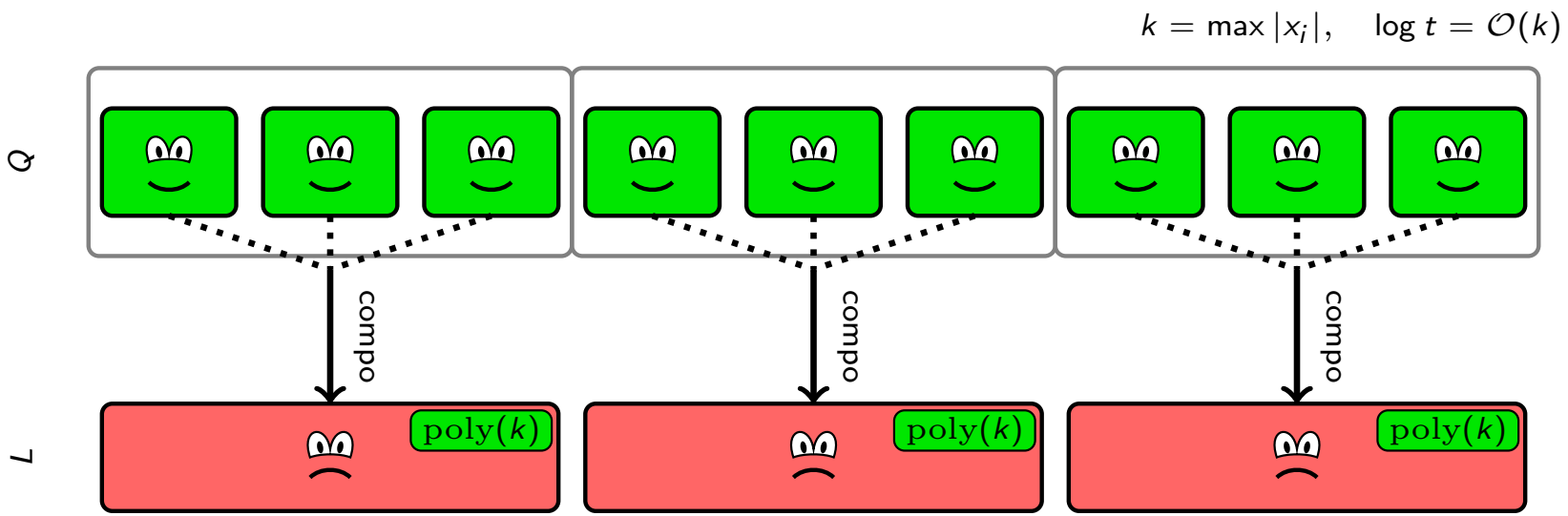
Proof

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

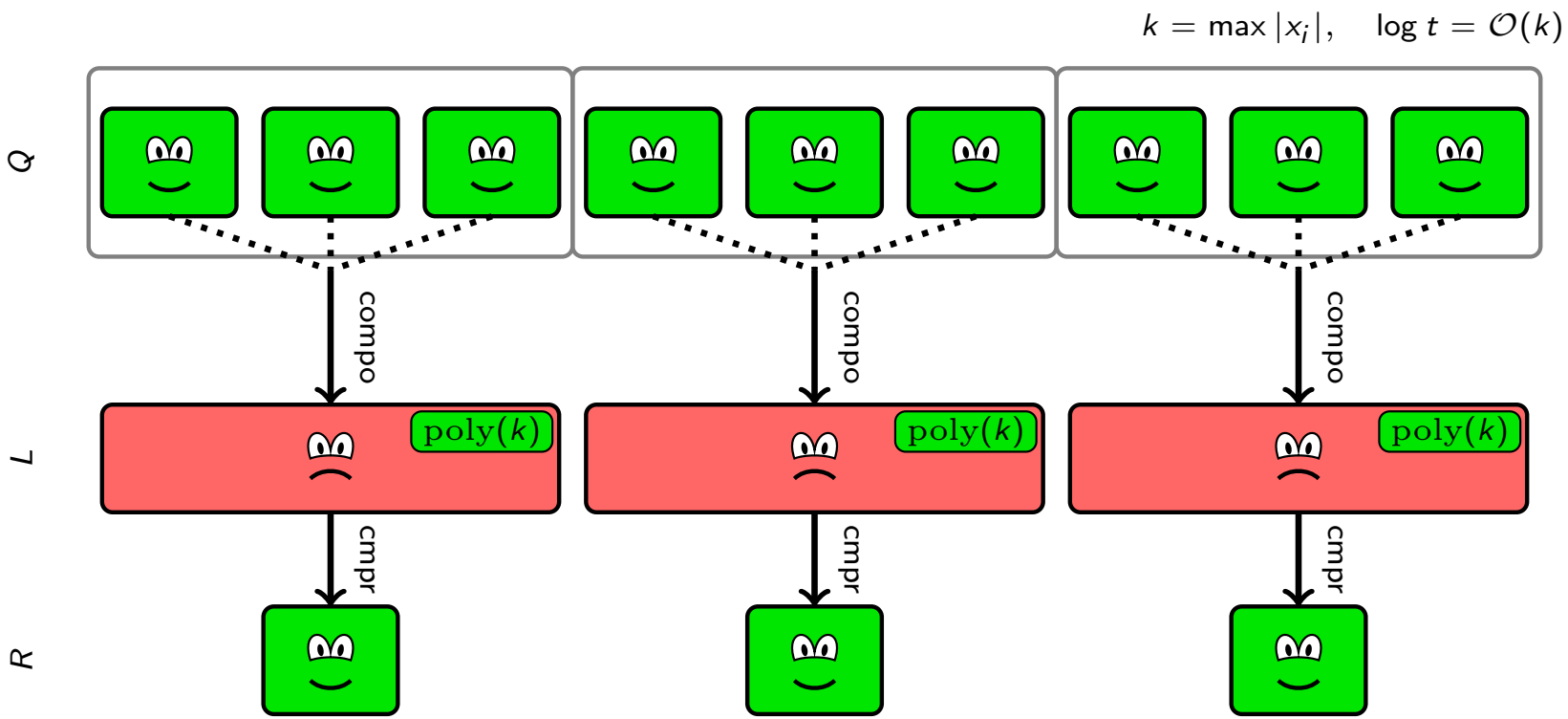
Q



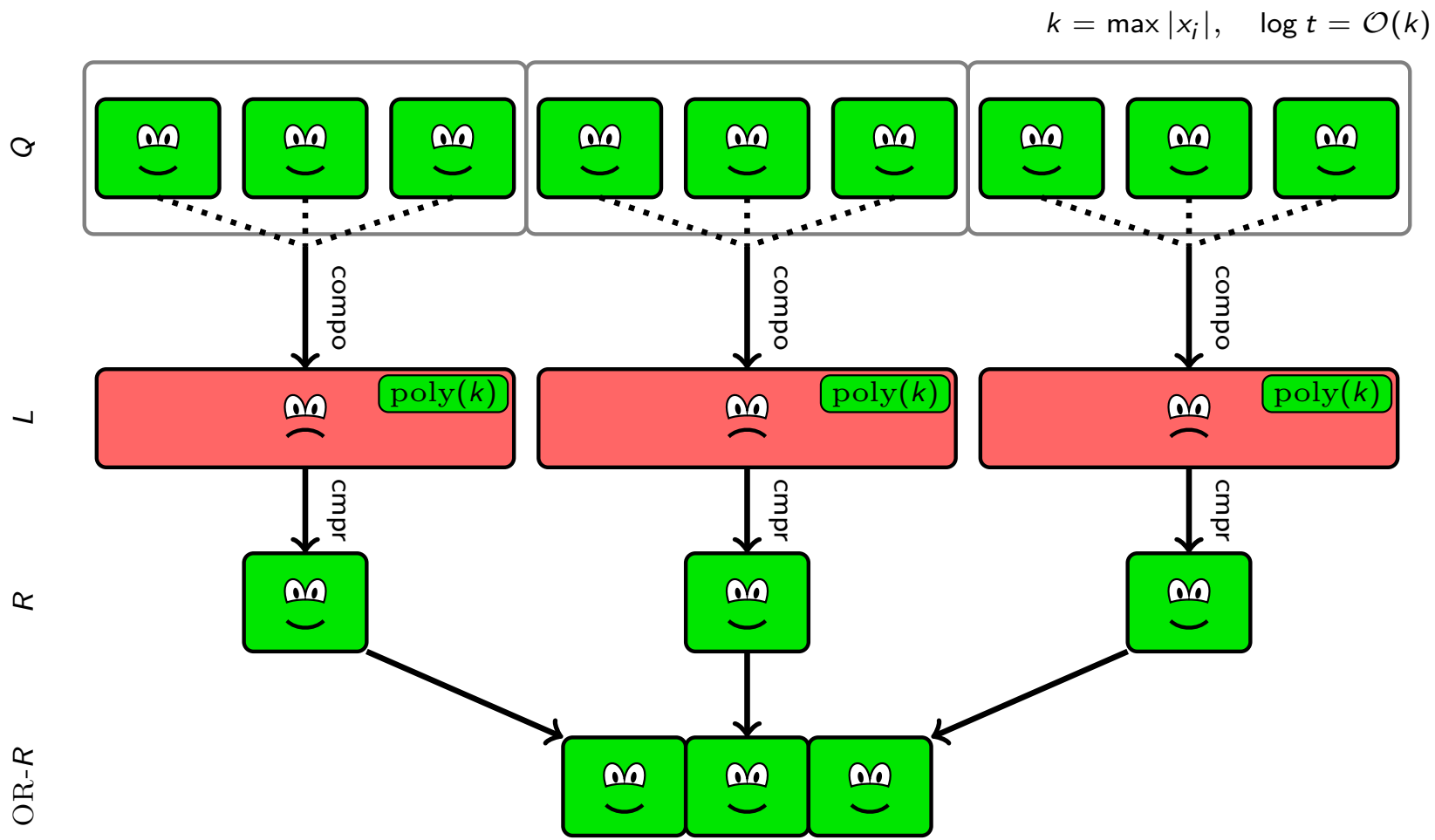
Proof


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

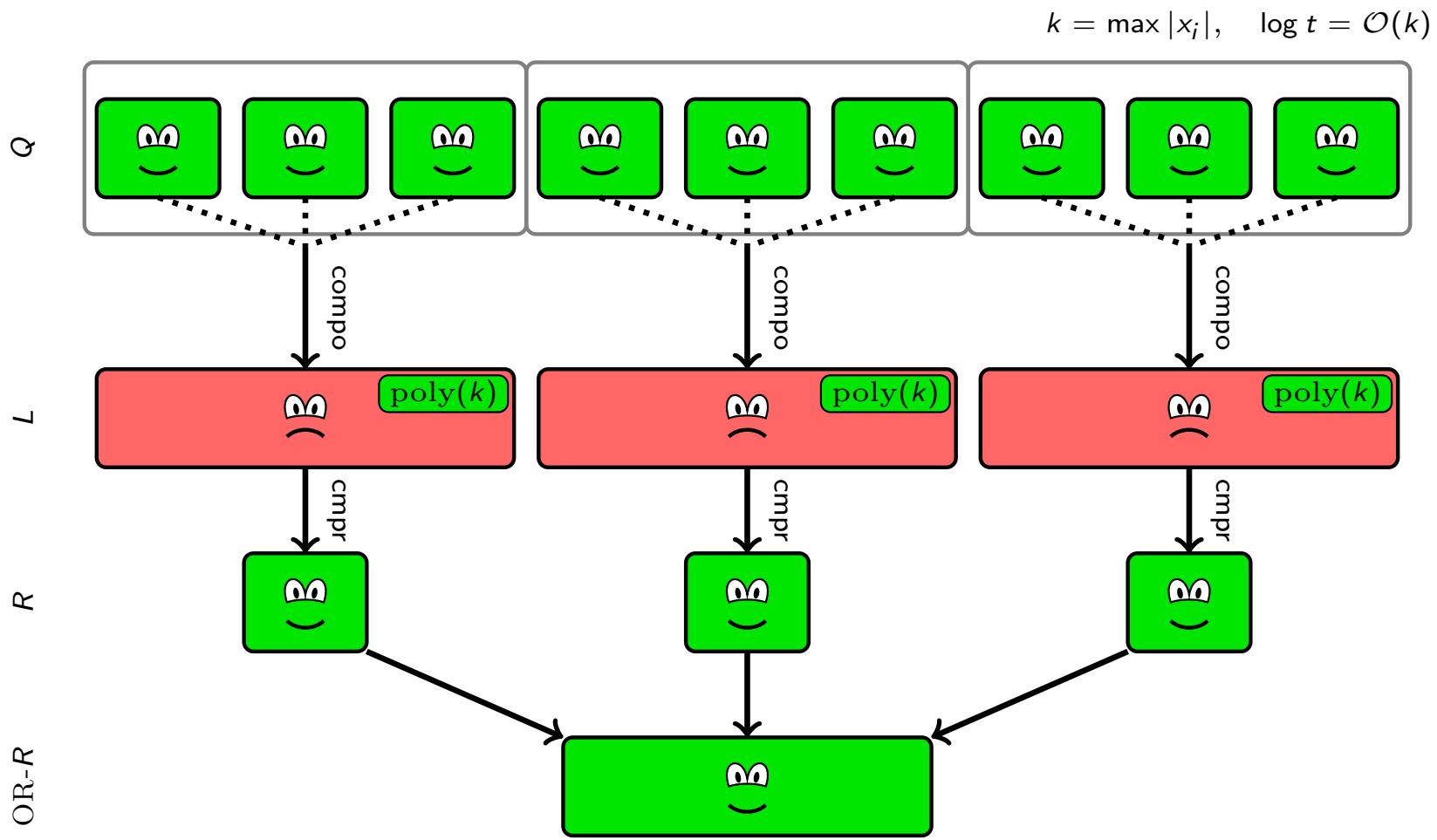
Proof


$k = \max |x_i|, \quad \log t = \mathcal{O}(k)$
Q
L
R
compo
compo
compo
poly(k)
poly(k)
poly(k)
cmpr
cmpr
cmpr

Proof


$k = \max |x_i|, \quad \log t = \mathcal{O}(k)$
Q
L
R
OR-R
compo
compo
compo
poly(k)
poly(k)
poly(k)
cmpr
cmpr
cmpr

Proof


$k = \max |x_i|, \quad \log t = \mathcal{O}(k)$
Q
L
R
OR-R
compo
compo
compo
poly(k)
poly(k)
poly(k)
cmpr
cmpr
cmpr

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 a few cross-compositions and give intuition about basic tricks.

Application 1: Set Splitting

Set Splitting

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

Parameter: $|U|$

Question: Does there exist a colouring $\mathcal{C} : U \to \{\mathbf{B}, \mathbf{W}\}$ 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

Input: Universe U and family of subsets $\mathcal{F} \subseteq 2^U$
Parameter: $|U|$
Question: Does there exist a colouring $\mathcal{C} : U \to \{\mathbf{B}, \mathbf{W}\}$ 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

Input: Universe U and family of subsets $\mathcal{F} \subseteq 2^U$
Parameter: $|U|$
Question: Does there exist a colouring $\mathcal{C} : U \to \{\mathbf{B}, \mathbf{W}\}$ 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

Input: Universe U and family of subsets $\mathcal{F} \subseteq 2^U$
Parameter: $|U|$
Question: Does there exist a colouring $\mathcal{C} : U \to \{\mathbf{B}, \mathbf{W}\}$ 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

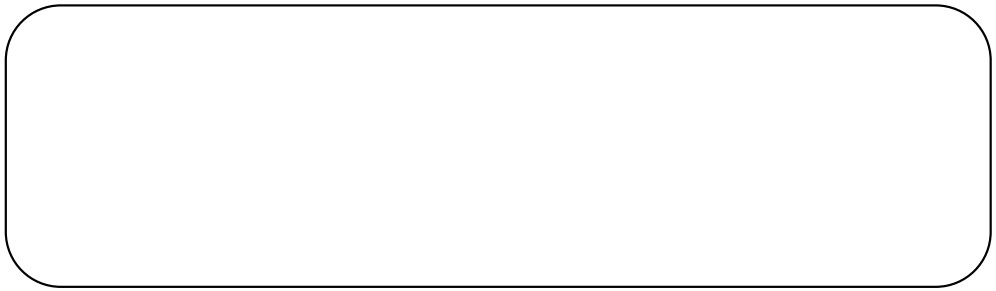
Input: Instances $(U, \mathcal{F}^i)$

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

Cross-composing into Set Splitting

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

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



PLAYGROUND

joint universe U

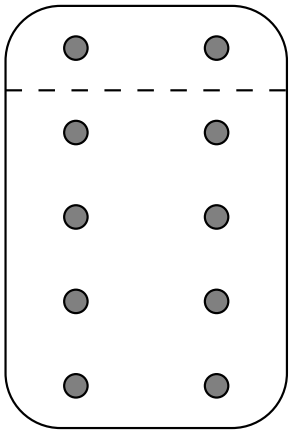
Cross-composing into Set Splitting

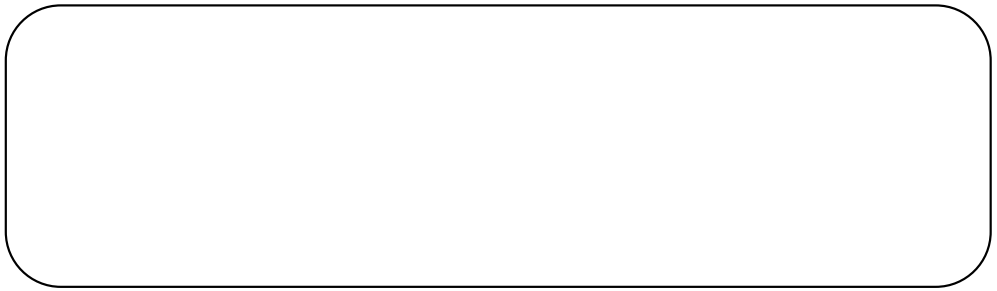
Input: Instances $(U, \mathcal{F}^i)$

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

INSTANCE SELECTOR

$1 + \log t$ pairs of vertices





PLAYGROUND

joint universe U

Cross-composing into Set Splitting

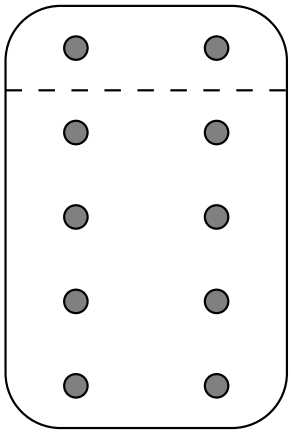
Input: Instances $(U, \mathcal{F}^i)$

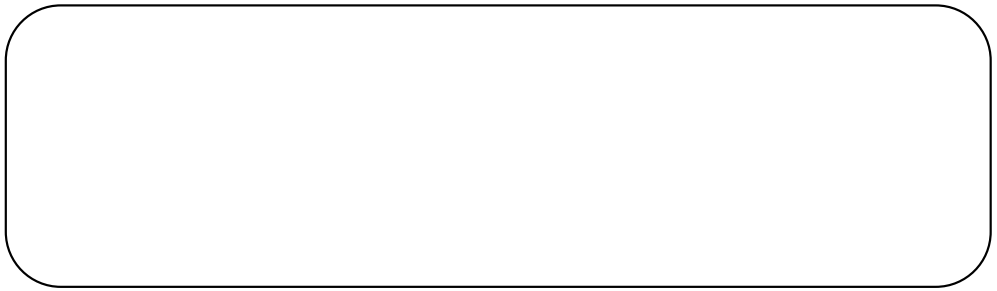
Output: Instance $(U^*, \mathcal{F}^*)$

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

INSTANCE SELECTOR

$1 + \log t$ pairs of vertices





PLAYGROUND

joint universe U

Cross-composing into Set Splitting

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

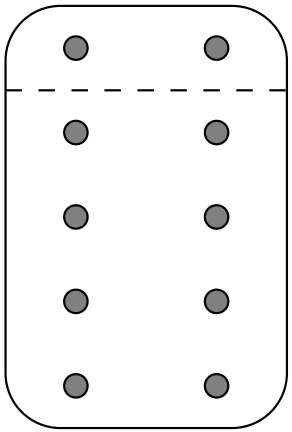
Output: Instance $(U^*, \mathcal{F}^*)$

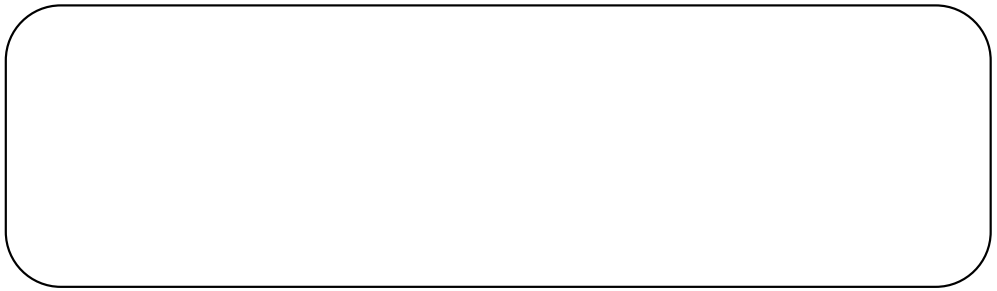
$|U^*| = |U| + 2\log t + 2$

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

INSTANCE SELECTOR

$1 + \log t$ pairs of vertices





PLAYGROUND

joint universe U

Cross-composing into Set Splitting

INSTANCE SELECTOR

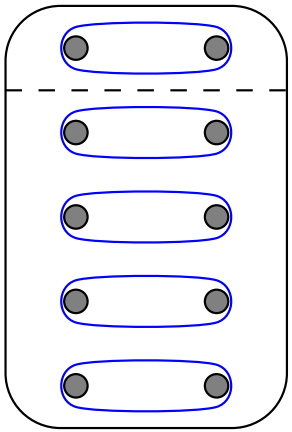
$1 + \log t$ pairs of vertices

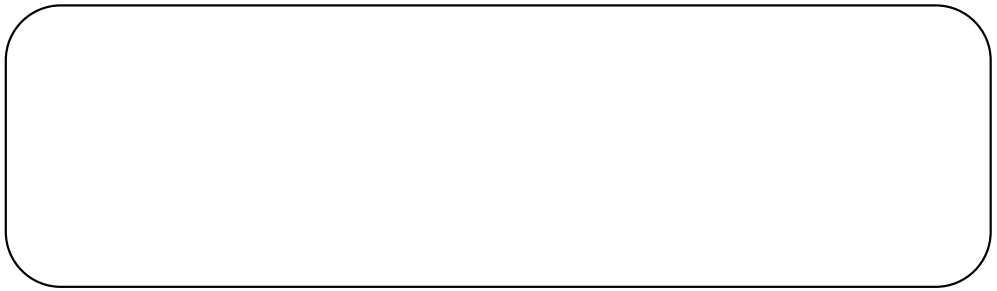
Input: Instances $(U, \mathcal{F}^i)$

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

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

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





PLAYGROUND

joint universe U

Cross-composing into Set Splitting

INSTANCE SELECTOR

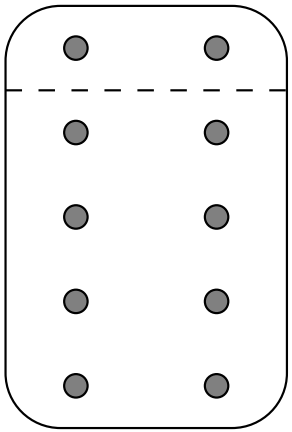
$1 + \log t$ pairs of vertices

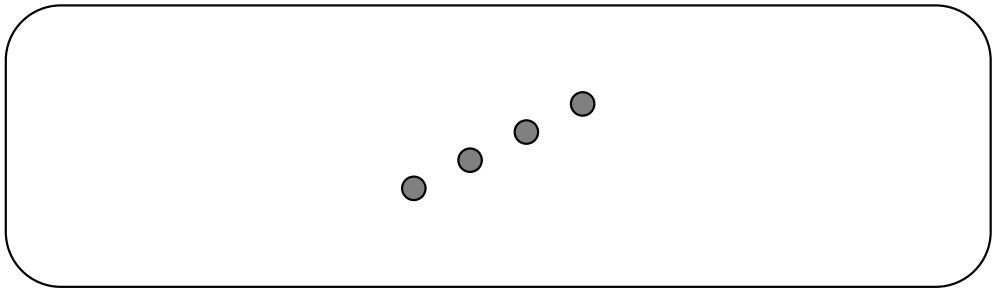
Input: Instances $(U, \mathcal{F}^i)$

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^*





PLAYGROUND

joint universe U

Cross-composing into Set Splitting

INSTANCE SELECTOR

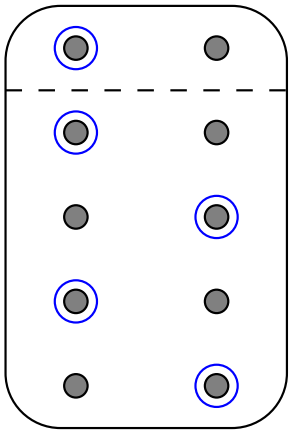
$1 + \log t$ pairs of vertices

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

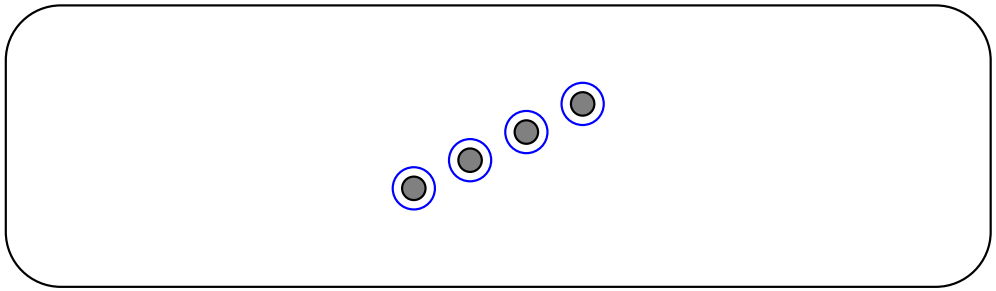
Output: Instance $(U^*, \mathcal{F}^*)$

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

$\mathcal{F}^*$ **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 guy, and binary encoding of i in IS



PLAYGROUND

joint universe U

Cross-composing into Set Splitting

INSTANCE SELECTOR

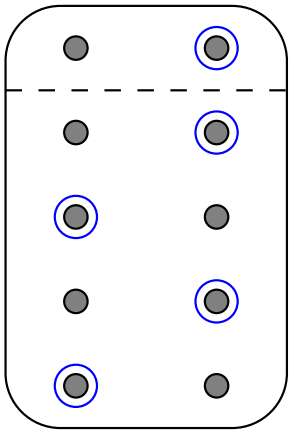
$1 + \log t$ pairs of vertices

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

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

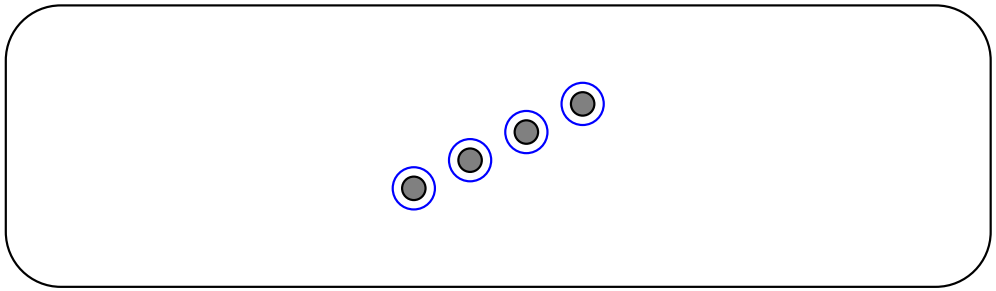
$|U^*| = |U| + 2\log t + 2$

$\mathcal{F}^*$ **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 guy, and binary encoding of i in IS

X_1^*: reverse X_0^* on IS



PLAYGROUND

joint universe U

Cross-composing into Set Splitting

INSTANCE SELECTOR

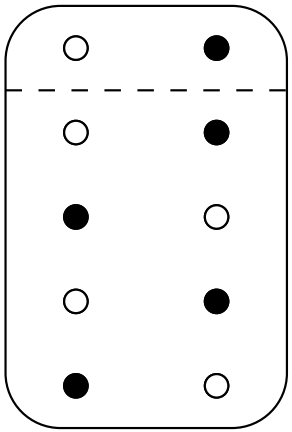
$1 + \log t$ pairs of vertices

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

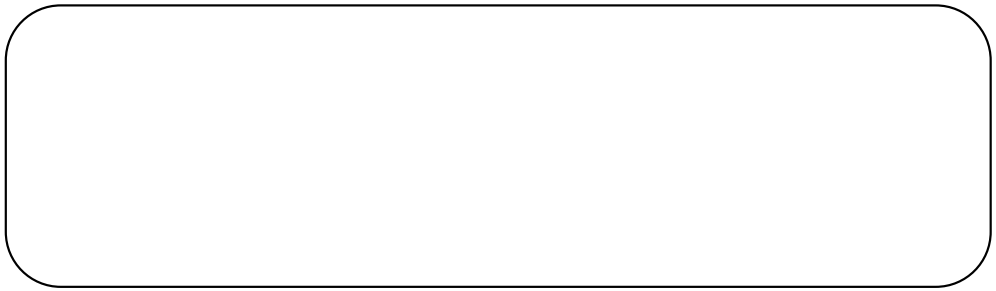
Output: Instance $(U^*, \mathcal{F}^*)$

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

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



Take any solution $\mathcal{C}$



PLAYGROUND

joint universe U

Cross-composing into Set Splitting

INSTANCE SELECTOR

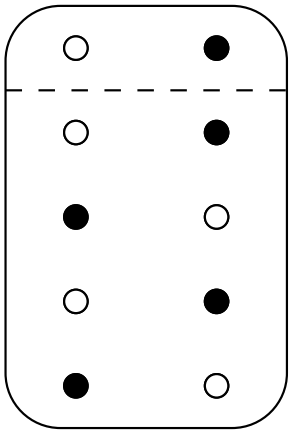
$1 + \log t$ pairs of vertices

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

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

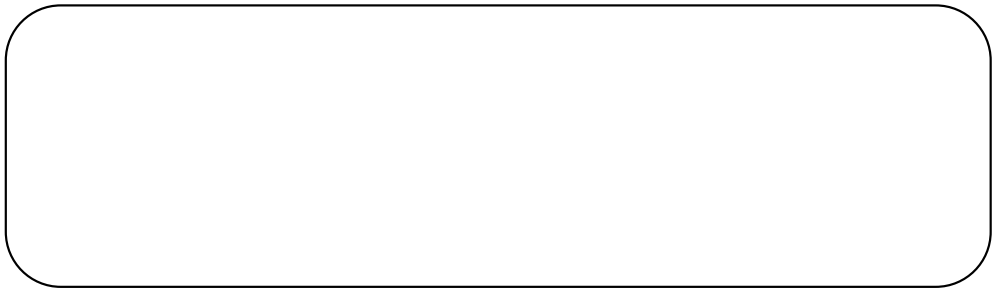
$|U^*| = |U| + 2 \log t + 2$

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



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

INSTANCE SELECTOR

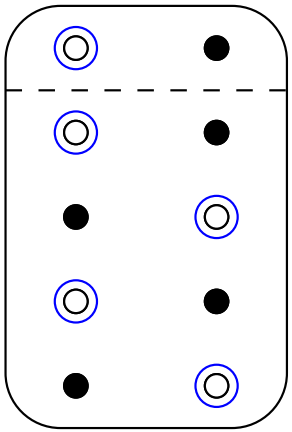
$1 + \log t$ pairs of vertices

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

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

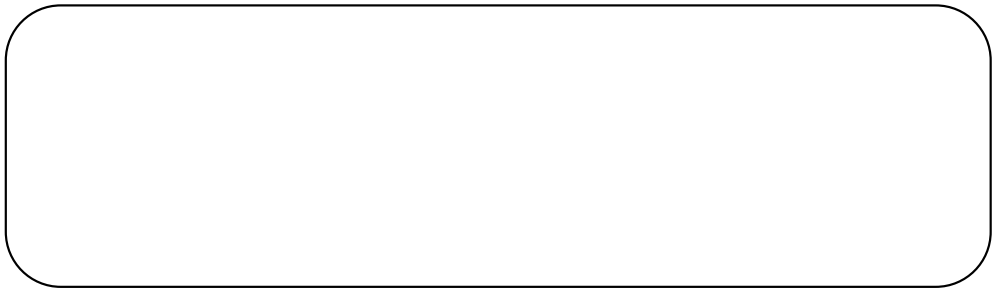
$|U^*| = |U| + 2\log t + 2$

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



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

INSTANCE SELECTOR

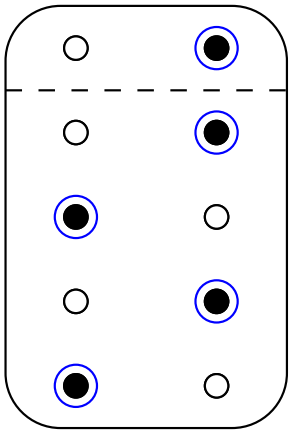
$1 + \log t$ pairs of vertices

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

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

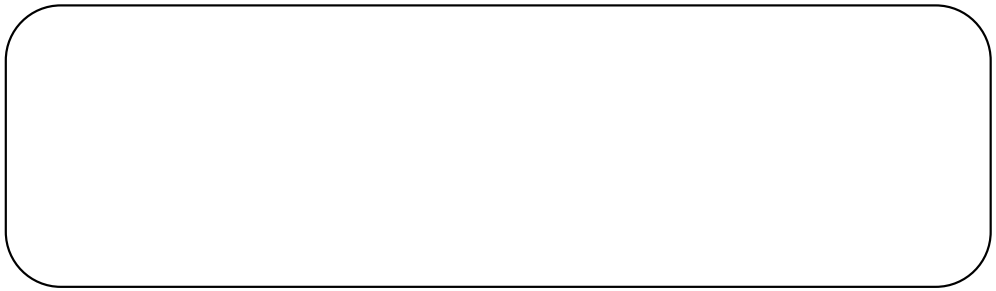
$|U^*| = |U| + 2 \log t + 2$

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



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

INSTANCE SELECTOR

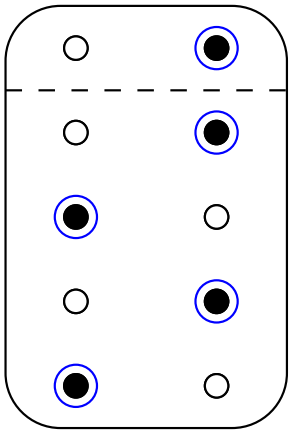
$1 + \log t$ pairs of vertices

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

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

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

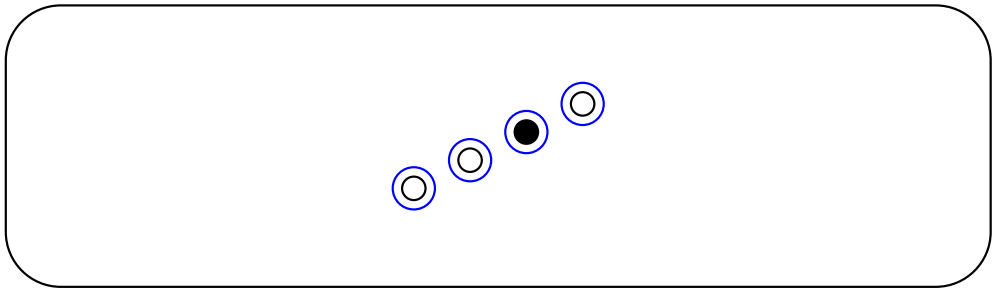
$\mathcal{F}^*$ **consists of**:
$1 + \log t$ 2-element sets for pairs,
$\forall X \in \mathcal{F}^i$, two sets X_0^*, X_1^*



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



PLAYGROUND

joint universe U

Cross-composing into Set Splitting

INSTANCE SELECTOR

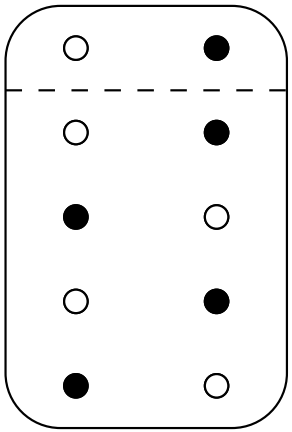
$1 + \log t$ pairs of vertices

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^*



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.

PLAYGROUND

joint universe U

Wrap-up

- Unparameterized Set Splitting cross-composes into Set Splitting parameterized by $|U|$.

Wrap-up

- Unparameterized Set Splitting cross-composes into Set Splitting parameterized by $|U|$.
- Unparameterized Set Splitting is **NP**-hard.

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.

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**:

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.

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.
  - One strategy is to choose $\log t$ bits of its index on an appropriate gadget.

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.
  - One strategy is to choose $\log t$ bits of its index on an appropriate gadget.
  - Choice of the index make the instance active, while the other instances are 'switched off'.

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.
  - One strategy is to choose $\log t$ bits of its index on an appropriate gadget.
  - Choice of the index make the instance active, while the other instances are 'switched off'.
- **Tomorrow**: More combinatorial examples.

AND-compositions

- Everything we said so far would work in the same manner for AND function instead of OR.

AND-compositions

- Everything we said so far would work in the same manner for AND function instead of OR.
- **Problem**: The proof of Fortnow and Santhanam inherently breaks for AND.

AND-compositions

- Everything we said so far would work in the same manner for AND function instead of OR.
- **Problem**: The proof of Fortnow and Santhanam inherently breaks for AND.

AND-distillation theorem Drucker; FOCS 2012

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

AND-compositions

- Everything we said so far would work in the same manner for AND function instead of OR.
- **Problem**: The proof of Fortnow and Santhanam inherently breaks for AND.

AND-distillation theorem Drucker; FOCS 2012

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

- All the rest of the framework works the same (AND-cross-compositions, etc.).

AND-compositions

- Everything we said so far would work in the same manner for AND function instead of OR.
- **Problem**: The proof of Fortnow and Santhanam inherently breaks for AND.

AND-distillation theorem Drucker; FOCS 2012

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

- All the rest of the framework works the same (AND-cross-compositions, etc.).
- In particular, Treewidth, Pathwidth, etc. do not admit polykernels, unless **NP** $\subseteq$ **coNP**$/\mathrm{poly}$.

Exercises

Exercise 15.4, points 1, 2, 11, 12, 13.