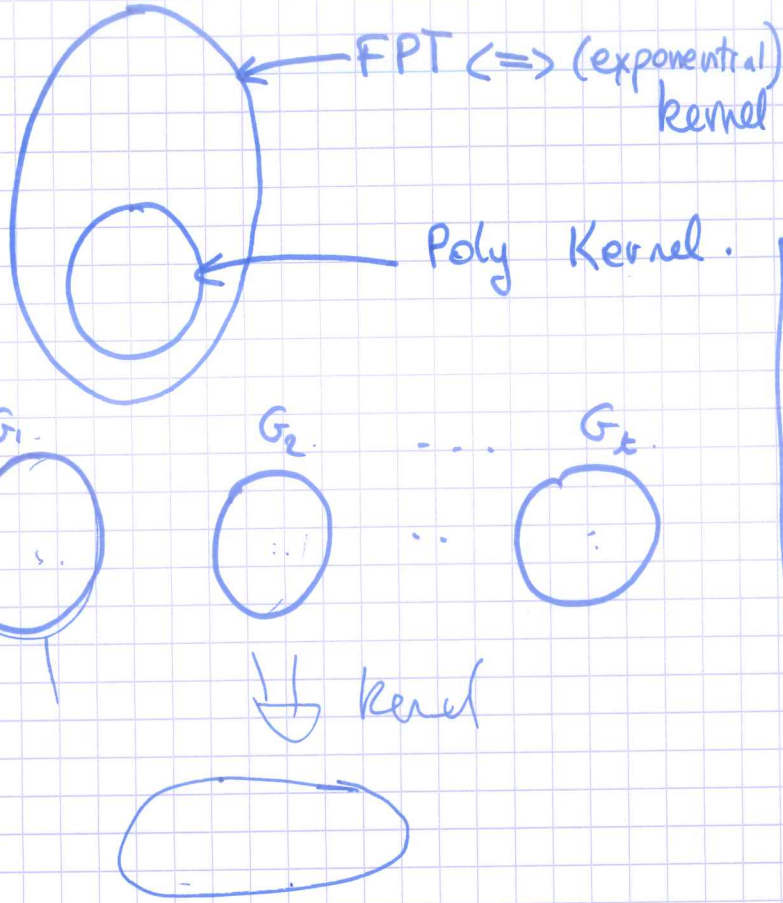
Kernel lower bounds



NP, coNP

↳ non-deterministic Algorithm

instance is negative ⇒ there is one computation path → negative answer

positive ⇒ all comput. path → positive answer

coNP/poly : a problem is in coNP/poly if there is a non-det. algorithm and a sequence of strings $(\alpha_n)_{n \in \mathbb{N}}$ (called advices)

$|\alpha_n| = poly(n)$

the algorithm can solve the problem co non-deterministically using the advice

Hypothesis for kernel lower bounds:

$NP \not\subseteq coNP/poly$.

Language point of view

Σ : alphabet of constant size (ex: $\{0,1\}$)

Σ^* : all words formed from Σ (= all instances)

- a problem is $L \subseteq \Sigma^*$ (positive instances)
- a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$
- kernel for Q: $(x,k) \xrightarrow{\text{P-time}} (x',k')$
  - $(x,k) \in Q \Leftrightarrow (x',k') \in Q$
  - $|x'| + k' \leq f(k)$ (f: computable function)

Distillations

Def: Let $L, R \subseteq \Sigma^*$. An OR-distillation of L into R is an algorithm which takes as input a sequence of instances $(x_1, \dots, x_t) \in \Sigma^*$, outputs $y \in \Sigma^*$ s.t.:

(i) runs in P-time $\left(\text{in } \sum_i |x_i|\right)$

(ii) $y \in R \Leftrightarrow \exists i \in \{1,\dots,t\}\ x_i \in L$

(iii) $|y| \leq p(\max_i |x_i|)$ for some poly p

Theorem If an NP-hard problem L has a OR-distillation to some problem R, then $NP \subseteq coNP/poly$.

Proof: if L has a distillation into R, then $L \in coNP/poly$. (what we want)

assume $\Sigma = \{0,1\}$

distillation

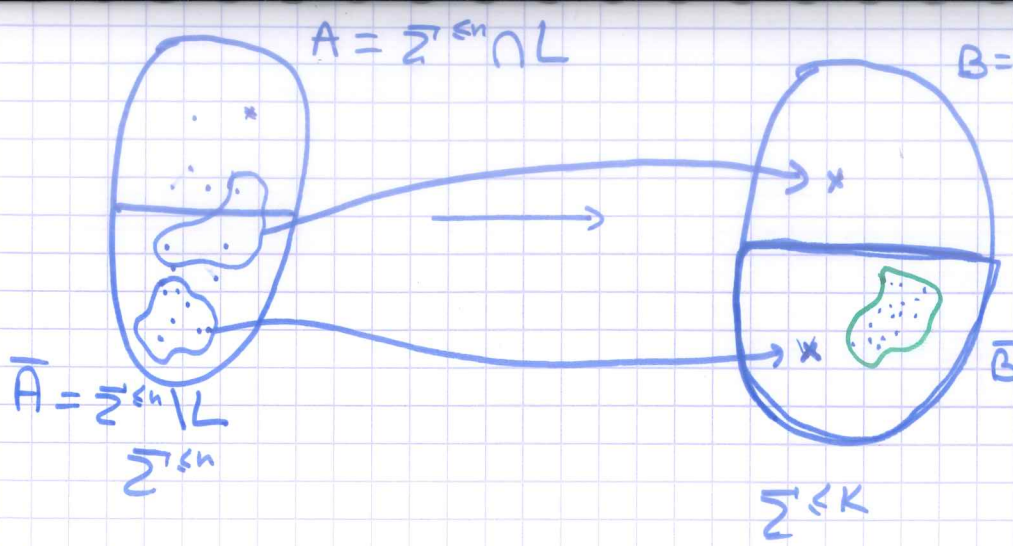
Let A be the distillation algo from L to some R

it maps instances of size at most n into an instance of size $p(n) =: K$.

$t = K+1$ instances.

$\Sigma^{\leq n}$: instances of size $\leq n$

$\Sigma^{\leq K}$: ——————— $\leq K$.



prove: $L \in$ coNP/poly

obvious advice: $\bar{B}$ but too large.

- $x \in \Sigma^{\leq n}$ is covered by $y \in \Sigma^{\leq K}$ if there is a t-tuple containing x $(x_1 .., x_t)$ s.t. $A(x_1 -, x_t) = y$
- $X \subseteq \Sigma^{\leq n}$ is covered by $Y \subseteq \Sigma^{\leq K}$ if every $x \in X$ is covered by some $y \in Y$

Claim: $\exists Y \subseteq \bar{B}$:

- $|Y| \leq n+1$
- $\bar{A}$ is covered by Y

How to use the claim?

string α_n encodes Y

($|Y|$ is polynomial in n, but $|y|$ is poly in n $\forall y \in Y$)

given $x \in \Sigma^*$: guess any possible t-tuple containing x $(x_1 .., x_t)$ (each $|x_i| \leq n$). Run $A(x_1 -, x_t) =: y$

- if $y \in Y$ $\Rightarrow$ answer NO
- if $y \notin Y$ $\Rightarrow$ answer YES

$\Big\}$ use advice α_n

- if $x \notin L$ $(\in \bar{A})$ $\Rightarrow$ there is a tuple which is sent to Y
- if $x \in L$ $(\in A)$ $\Rightarrow$ any tuple will be send to B $(\notin Y)$.

Proof of claim: construct $Y = \{y_1, \dots, y_q\}$ one by one $Y_i = \{y_1 \dots y_i\}$ $Y_0 = \emptyset$

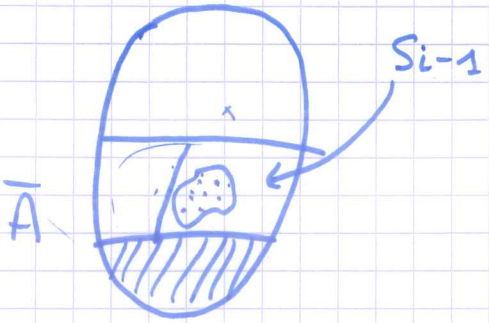
$\forall i$ $S_i \subseteq \bar{A}$: elements not covered by Y_i

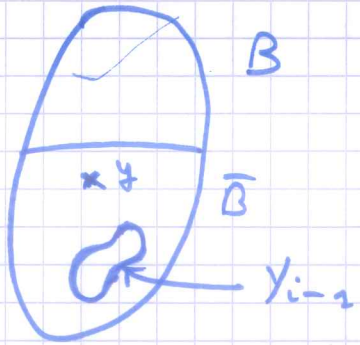
$S_0 = \bar{A}$

we will maintain $|S_i| \leq \frac{|S_{i-1}|}{2}$

since $|\bar{A}| \leq |\Sigma^*|^n \leq 2^{n+1}$, S_i will be become empty after $n+1$ steps.

assume we have computed Y_{i-1}





$|\bar{B}| \leq 2^{K+1}$ $\Rightarrow$ there is a $y \in \bar{B}$ covering $\frac{1}{2^{K+1}}$ fraction of $(S_{i-1})^t$

~~element~~ tuples now covered $|A^{-1}(y) \cap (S_{i-1})^t| \geq \frac{|(S_{i-1})^t|}{2^{k+1}}$

$\geq \left(\frac{|S_{i-1}|}{2}\right)^t$ S_i = elements not covered by $Y_{i-1} \cup \{y\}$

all strings in $A^{-1}(y) \cap (S_{i-1})^t$ are now covered $\Rightarrow A^{-1}(y) \cap (S_{i-1})^t \subseteq (S_{i-1} \setminus S_i)^t$

$|A^{-1}(y) \cap (S_{i-1})^t| \leq |(S_{i-1} \setminus S_i)^t| \leq |S_{i-1} \setminus S_i|^t$

$\Rightarrow |S_{i-1} \setminus S_i|^t \geq \left(\frac{|S_{i-1}|}{2}\right)^t$

$\Rightarrow |S_i| \leq \frac{|S_{i-1}|}{2}$ ■

How to use distillations for kernel lower bound?

Def: given a language L and a parameterized language $Q \subseteq \Sigma^* \times \mathbb{N}$. a cross-composition from L to Q is an algorithm A which takes as input $x_1, \ldots, x_t \in \Sigma^*$ and output $(y,k) \in \Sigma^* \times \mathbb{N}$: st:

(i) (y,k) is computed in P-time in $\sum |x_i|$

(ii) $(y,k) \in Q \iff \exists i \in \{1..t\}$ $x_i \in L$

(iii) $k \leq p(\max |x_i| + \log_2 t)$ for some polynomial p

⚠ difficulty: k does not depend linearly on t

a Polynomial equivalence relation R on Σ^* is an equivalence relation such that:

- $x_1 R x_2$ can be done in polynomial time
- R partitions the set of all instances of size $\leq n$ into at most $p(n)$ for some polynomial.

equivalent wrt polynomial equivalence relation

Theorem: if an NP-hard language L cross-composes into $Q \subseteq \Sigma^* \times \mathbb{N}$, then Q has no polynomial kernel unless $NP \subseteq coNP/poly$.

Proof: idea:

cross-composition + Poly kernel $\Rightarrow$ distillation.

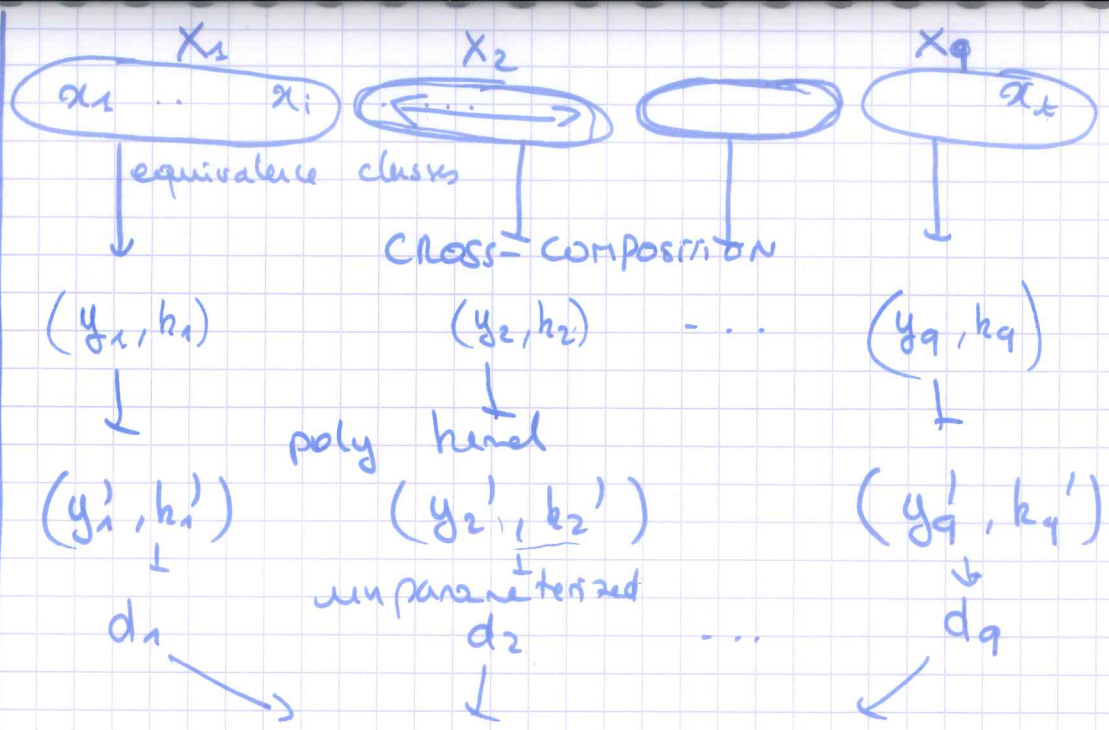
- given $Q \subseteq \Sigma^* \times \mathbb{N}$ $\rightarrow$ unparameterized version $\tilde{Q}$: append the parameter at the end of the string in unary.
- given $L \subseteq \Sigma^*$: $OR(L)$: each input is a sequence of instances, positive if one of the is positive. $x_1 \# x_2 \# \dots \# x_t$
- $Q \subseteq \Sigma^* \times \mathbb{N} \rightsquigarrow OR(\tilde{Q})$

---

in CROSS-COMPOSITION, we may always assume that $\log t = O(n)$.

$\Rightarrow$ you may remove duplicated instances

$\Rightarrow$ you have at most $|\Sigma|^n$ instances (distinct)



Claim: this is a distillation from L to $OR(\tilde{Q})$

- equivalence: d is positive $\Leftrightarrow \exists i$: d_i is pos.

$\Leftrightarrow \dots$

$\Leftrightarrow \exists i \in \{1, \dots, t\}$ x_i is positive.

SIZE

- $|d|$ has to be polynomial in $n = \max |x_i|$

q is polynomial in n ✓

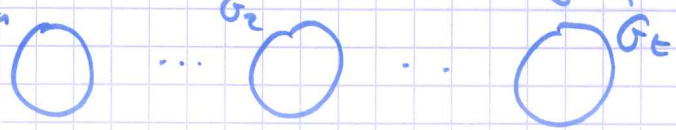
size of each d_i? size of (y_i', k_i')?

$|y_i'| + k_i' \leq poly(k_i)$

$k_i \leq poly(n + \log t) \leq poly(n)$

Some trivial cross-compositions: when disjoint union works

- most problems parameterized by the treewidth of the input graph.



$tw(\bigcup G_i) \leq \max_i tw(G_i)$

⇒ they don't admit PK param. by tw unless NP ⊆ coNP/poly

- Max ind. set
- Dominating set
- ⋮

Remarks: what about replacing OR by AND ? ⇒ NP ⊆ NP/poly (stupid)

AND-distillation unlikely for NP-hard problem ??

[Drucker 2004] → implies NP ⊆ coNP/poly