Three results about BPP

BPP is in P/poly (Adleman)
BPP is in the polynomial time hierarchy (Sipser-Gacs-Lautemann)
If NP is in BPP then the polynomial time hierarchy collapses (Karp-Lipton)

The Polynomial Time Hierarchy

- Σ_2^p: set of languages L for which there is a polynomial-time DTM M such that

  w is in L iff $\exists Z_1\ \forall Z_2\ M(w,Z_1,Z_2)$ accepts

  where $|Z_1| = |Z_2|$ is polynomial in $|w|$

- Π_2^p: similar to Σ_2^p, but starts with a $\forall$ quantifier

  w is in L iff $\forall Z_1\ \exists Z_2\ M(w,Z_1,Z_2)$ accepts
- Π_k^p and Σ_k^p: generalizations for k > 2
- Polynomial time hierarchy (PH) is $\cup_{k>0} (\Pi_k^p \cup \Sigma_k^p)$

A Hierarchy of Quantified SAT Problems

- $\sum_k$ SAT: set of true quantified formulas of the form

  $$\exists X_1 \, \forall X_2 \ldots Q_k X_k \, \phi(X_1, X_2, \ldots, X_k)$$

  where for some $n \geq 0$
  - ϕ is a Boolean formula over kn variables
  - $X_i = x_{i1}, \ldots, x_{in}$ for all i is a truth assignment to variables of ϕ
- $\sum_k$ SAT is complete for $\sum_k^p$

- $\prod_k$ SAT: similar, but starts with a $\forall$ quantifier
- $\prod_k$ SAT is complete for $\prod_k^p$

The Polynomial Time Hierarchy

Claim: If $\Pi_k^p \subseteq \Sigma_k^p$ then in fact $\Pi_k^p = \Sigma_k^p$.

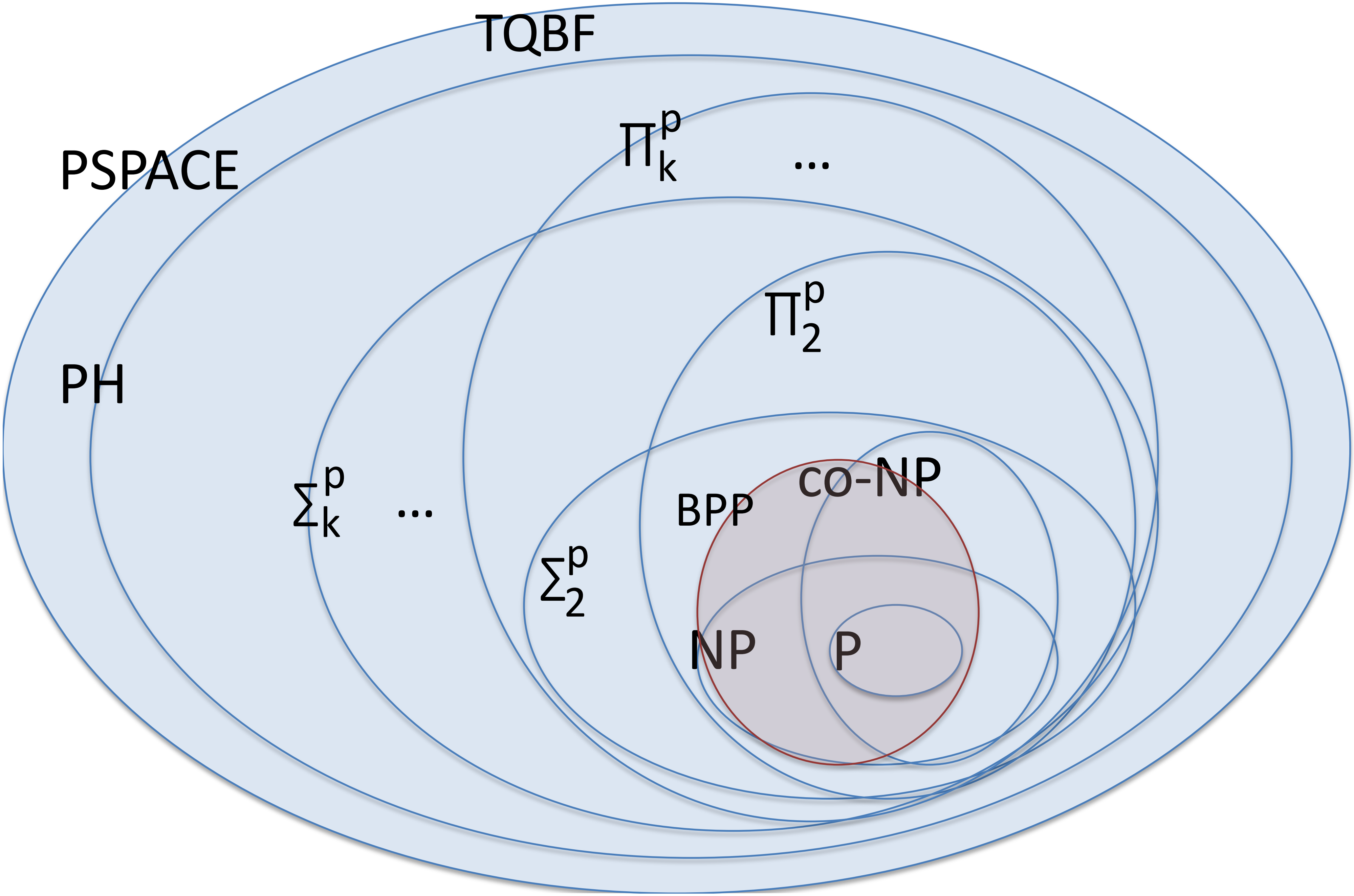
Proof: Let S be any set of languages, and let

$$\text{co-S} = \{ \overline{L} \mid L \text{ is in } S \}.$$

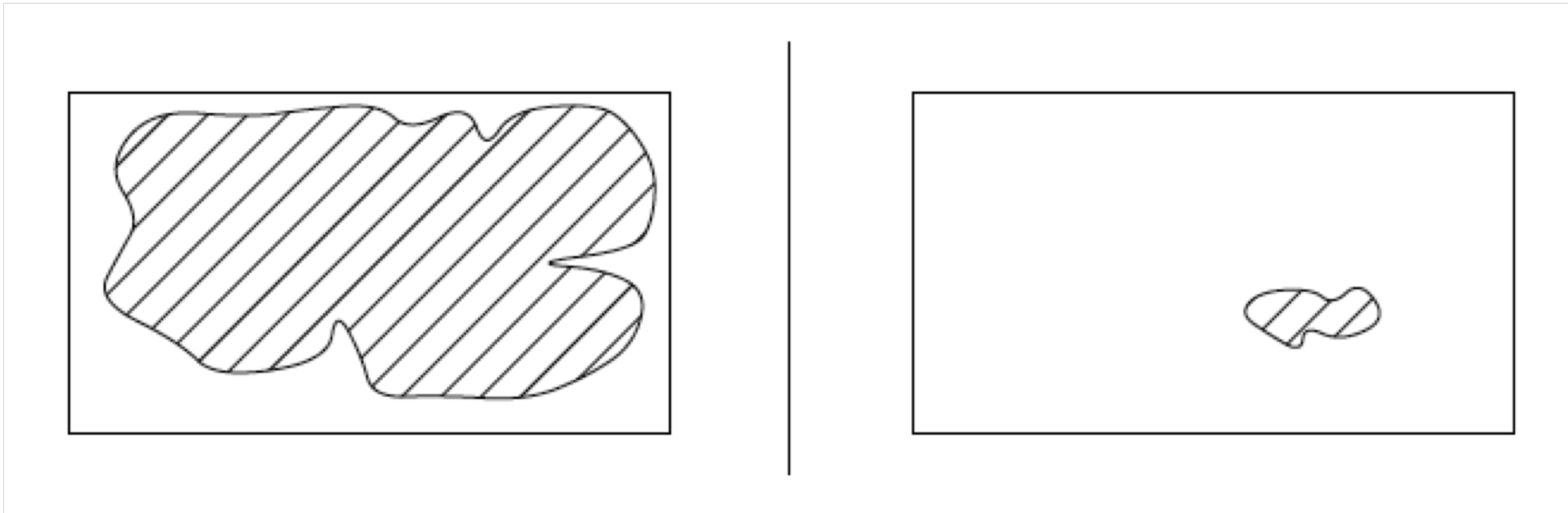
Suppose that co-S ⊆ S; we'll show that S ⊆ co-S, and so S = co-S.

Let L be in S. Since co-S ⊆ S, $\overline{L}$ must also be in S. And then, since $\overline{L}$ is in S, L must be in co-S. So S ⊆ co-S.

BPP and the Polynomial Time Hierarchy



BPP is Contained in $\Sigma_2^p \cap \Pi_2^p$



- Illustration of sets S_w of coin flip sequences r such that M accepts *w* on r. Picture on the left is when *w* is in L, and picture on the right is when *w* is not in L.

From Arora-Barak textbook

If NP is in BPP then PH Collapses

If NP is in BPP then PH Collapses

- "Collapses" means that PH is contained in Σ_2^p
- The proof in two parts:
  a) If NP is contained in P/poly then $\Pi_2^p \subseteq \Sigma_2^p$
  b) If $\Pi_2^p \subseteq \Sigma_2^p$ then every language in PH is in Σ_2^p

If NP is in BPP then PH Collapses

- Suppose that NP is contained in P/poly
- There is a DTM, say M'', and a poly-length advice sequence {A(n)} such that, given an instance w of SAT,

  w is in SAT iff M'' accepts on input w, advice A(|w})

If NP is in BPP then PH Collapses

- Suppose that NP is contained in P/poly
- There is a DTM, say M'', and a poly-length advice sequence {A(n)} such that, given an instance w of SAT,

  w is in SAT iff M'' accepts on input w, advice A(|w})

- There is a DTM, say M', and a poly-length advice sequence {A'(n)} such that, given an instance w of SAT, w is in SAT iff M' outputs a satisfying truth assignment for w

If NP is in BPP then PH Collapses

- "Collapses" means that PH is contained in Σ_2^p
- The proof in two parts:
  a) If NP is contained in P/poly then $\Pi_2^p \subseteq \Sigma_2^p$
  b) If $\Pi_2^p \subseteq \Sigma_2^p$ then every language in PH is in Σ_2^p

Summary

- BPP is "low" in the polynomial time hierarchy, deep within PSPACE
- Also, BPP is unlikely to contain NP: if NP is in BPP then the PH collapses
  - "if pigs could whistle then horses could fly" type of result
  - to prove this, an unusual complexity class came in handy: P/poly

Space Bounded Randomized Complexity Classes

one-sided error, log space bounded classes

handy techniques for probabilistic reasoning

RL

- A language L is in RL if there is an O(log n)-space PTM M such that
  - if $x \in L$ then Pr[M accepts x] $\geq 2/3$ and
  - if $x \notin L$ then Pr[M accepts x] $= 0$

NL = RL

- How to design a randomized, log-space algorithm for PATH = { (G,s,t) | node t can be reached from node s in directed graph G}?
- Throughout, let G = (V,E) where V = {1,2, ..., n} and let e = |E|

NL = RL

- How to design a randomized, log-space algorithm for PATH = { (G,s,t) | node t can be reached from node s in directed graph G}?

- Idea: Repeatedly follow a random path from s, and accept iff one of the paths reaches t
- Random path from s: visit s initially and when node i is visited, choose an adjacent node j uniformly at random to visit next

A Randomized, Log-Space Algorithm for PATH

Without loss of generality, suppose that each node of G has at most two children

Repeat

- Follow a random path from s until either
  - t is reached: halt and accept
  - a dead end is reached, or n-1 steps have been taken
- Flip n+2 random coins, halt and reject if all are heads

RLP

- PATH is in RL because a log space probabilistic TM can run for exponential expected time, using “probabilistic counting”

- A language L is in RLP if there is an log-space *and poly-time* PTM M such that
  - if $x \in L$ then Pr[M accepts x] $\geq 2/3$ and
  - if $x \notin L$ then Pr[M accepts x] $= 0$

UPATH is in RLP

- UPATH = { (G,s,t) | node t can be reached from node s in an *undirected* graph G}

UPATH is in RLP

UPATH Algorithm:

- On input (G,s,t), follow a random path from s
  - If t is reached at some step, halt and accept
  - If t is not reached within 6e(n-1) steps, halt and reject

- The algorithm is correct if t is not reachable from s, since it must reject

- What if t is reachable from s? (To be continued next time)