QSAT is PSPACE-complete

Theorem: QSAT is PSPACE-complete.

- Proof:
  - in PSPACE:
    - \(3x_1 \exists x_2 \exists x_3 \ldots \exists x_n \varphi(x_1, x_2, \ldots, x_n)?\)
  - "3x_1": for each \(x_1\), recursively solve
    \(\exists x_2 \exists x_3 \ldots \exists x_n \varphi(x_1, x_2, \ldots, x_n)?\)
  - if encounter "yes", return "yes"
  - "\(\forall x_1\)": for each \(x_1\), recursively solve
    \(\exists x_2 \exists x_3 \ldots \exists x_n \varphi(x_1, x_2, \ldots, x_n)?\)
  - if encounter "no", return "no"
  - base case: evaluating a 3-CNF expression
  - poly(n) recursion depth
  - poly(n) bits of state at each level
QSAT is **PSPACE**-complete

- Key observation #1:
  
  \[
  \exists Z [ \text{REACH}(A, Z, i) \land \text{REACH}(Z, B, i)]
  \]

  - cannot define \( \psi_{i+1}(A, B) \) to be
    \[
    \exists Z [ \psi_i(A, Z) \land \psi_i(Z, B)]
    \]
  (why?)

QSAT is **PSPACE**-complete

- Key idea #2: use quantifiers

  - couldn't do \( \psi_{i+1}(A, B) = \exists Z [ \psi_i(A, Z) \land \psi_i(Z, B)] \)
  - define \( \psi_{i+1}(A, B) \) to be
    \[
    \exists Z \exists X \exists Y [ (X=A \land Y=Z) \lor (X=Z \land Y=B)] \psi_i(X, Y)
    \]
  - \( \psi_i(X, Y) \) is preceded by quantifiers
  - move to front (they don’t involve \( X,Y,Z,A,B \))

\[ \psi_0(A, B) = \text{true iff } A = B \text{ or } A \text{ yields } B \text{ in 1 step} \]

\[ \psi_{i+1}(A, B) = \exists Z [ \psi_i(A, Z) \land \psi_i(Z, B)] \]

  - total size of \( \psi_{i+n} \) is \( O(n^k) \)
  - logspace reduction

**PH collapse**

**Theorem:** if \( \Sigma_i = \Pi_i \) then for all \( j > i \)

\[ \Sigma_j = \Pi_j = \Delta_j = \Sigma_i \]

“the polynomial hierarchy collapses to the i-th level”

- Proof:
  - sufficient to show \( \Sigma_i = \Sigma_{i+1} \)
  - then \( \Sigma_{i+1} = \Pi_i = \Pi_{i+1} \); apply theorem again

**Oracles vs. Algorithms**

A point to ponder:

- given poly-time algorithm for SAT
  - can you solve MIN CIRCUIT efficiently?
  - what other problems? Entire complexity classes?

- given SAT oracle
  - same input/output behavior
  - can you solve MIN CIRCUIT efficiently?
Natural complete problems

• We now have versions of SAT complete for levels in PH, PSPACE

• Natural complete problems?
  – PSPACE: games
  – PH: almost all natural problems lie in the second and third level

Natural complete problems in PH

– MIN CIRCUIT
  • good candidate to be $\Sigma_2$-complete, still open

– MIN DNF: given DNF $\phi$, integer k; is there a DNF $\phi'$ of size at most k computing same function $\phi$ does?

**Theorem** (U): MIN DNF is $\Sigma_2$-complete.

Natural complete problems in PSPACE

• General phenomenon: many 2-player games are PSPACE-complete.
  – 2 players I, II
  – alternate picking edges
  – lose when no unvisited choice

  GEOGRAPHY = {$(G, s)$ : G is a directed graph and player I can win from node s}

**Theorem**: GEOGRAPHY is PSPACE-complete.

**Proof**:
  – in PSPACE
    • easily expressed with alternating quantifiers
  – PSPACE-hard
    • reduction from QSAT

Natural complete problems in PSPACE

• $9 \times 1$

  \[
  \begin{array}{ccc}
  \text{true} & \text{false} & \text{true} \\
  \text{true} & \text{false} & \text{false} \\
  \text{true} & \text{true} & \text{false} \\
  \text{true} & \text{false} & \text{true} \\
  \end{array}
  \]

Karp-Lipton

• we know that $P = NP$ implies SAT has polynomial-size circuits.
  – (showing SAT does not have poly-size circuits is one route to proving $P \neq NP$)

• suppose SAT has poly-size circuits
  – any consequences?
  – might hope: SAT $\in P/poly \Rightarrow$ PH collapses to P, same as if SAT $\in P$
Karp-Lipton

**Theorem (KL):** if SAT has poly-size circuits then \( \text{PH} \) collapses to the second level.

- **Proof:**
  - suffices to show \( \Pi_2 \subseteq \Sigma_2 \)
  - \( L \in \Pi_2 \) implies \( L \) expressible as:
    \[
    L = \{ x : \forall y \exists z (x, y, z) \in R \}
    \]
    with \( R \in \text{P} \).

---

Karp-Lipton

\[
L = \{ x : \forall y \exists z (x, y, z) \in R \}
\]

\[
\{ x : \exists C \forall y \ [\text{use } C \text{ repeatedly to find some } z \text{ for which } (x, y, z) \in R; \text{ accept iff } (x, y, z) \in R] \}
\]

---

BPP \( \subseteq \) PH

- **Recall:** don’t know BPP different from EXP

**Theorem (S,L,GZ):** \( \text{BPP} \subseteq (\Pi_2 \setminus \Sigma_2) \)

- don’t know \( \Pi_2 \setminus \Sigma_2 \) different from EXP but believe much weaker

---

BPP \( \subseteq \) PH

- **Proof:**
  - BPP language \( L \): p.p.t. TM M:
    \[
    x \in L \Rightarrow \Pr_y[M(x,y) \text{ accepts}] \geq \frac{2}{3}
    \]
    \[
    x \notin L \Rightarrow \Pr_y[M(x,y) \text{ rejects}] \geq \frac{2}{3}
    \]
  - strong error reduction: p.p.t. TM \( M' \)
    - use \( n \) random bits \( \{y\} = n \)
    - \# strings \( y' \) for which \( M'(x, y') \) incorrect is at most \( 2^{-\frac{3}{2}} \)
    - (can’t achieve with naïve amplification)

---

BPP \( \subseteq \) PH

- view \( y' = (w, z) \), each of length \( n/2 \)
- consider output of \( M'(x, (w, z)) \):
  - view \( y' = (w, z) \), each of length \( n/2 \)
  - consider output of \( M'(x, (w, z)) \):
  - view \( y' = (w, z) \), each of length \( n/2 \)
  - consider output of \( M'(x, (w, z)) \):
BPP \subseteq \text{PH}

• proof (continued):
  – strong error reduction: \# bad \ y' < 2^{n/3}
  – \ y' = (w, z) with \ |w| = |z| = n/2
  – Claim: \ L = \{x : \exists w \forall z \ M'(x, (w, z)) = 1 \}
  – \ x \in L: \text{ suppose } \forall w \exists z \ M'(x, (w, z)) = 0
     • implies \ 2^{\Omega(n)} \text{ 0's; contradiction}
  – \ x \notin L: \text{ suppose } \exists w \forall z \ M'(x, (w, z)) = 1
     • implies \ 2^{\Omega(n)} \text{ 1's; contradiction}

\text{BPP} \subseteq \text{PH}

– given \text{BPP} language \ L: \text{p.p.t. TM } M:
  \ x \in L \Rightarrow \Pr_y[M(x,y) \text{ accepts}] \geq 2/3
  \ x \notin L \Rightarrow \Pr_y[M(x,y) \text{ rejects}] \geq 2/3
– showed \ L = \{x : \exists w \forall z M'(x, (w, z)) = 1 \}
– thus \text{BPP} \subseteq \Sigma_2
– \text{BPP} closed under complement \Rightarrow \text{BPP} \subseteq \Pi_2
– conclude: \text{BPP} \subseteq (\Pi_2 \cap \Sigma_2)

New Topic

The complexity of \text{counting}

Counting problems

• So far, we have ignored \text{function problems}
  – given \ x, compute \( f(x) \)

• important class of \text{function problems}:
  \text{counting problems}

  – e.g. given 3-CNF \( \phi \) how many satisfying assignments are there?

Counting problems

• \#P is the class of functional problems expressible as:
  \begin{align*}
  \text{input } x & : f(x) = |\{y : (x, y) \in R\}| \\
  \text{where } R & \in \text{P}.
  \end{align*}

• compare to \textbf{NP} (decision problem)
  \begin{align*}
  \text{input } x & : f(x) = \exists y : (x, y) \in R ? \\
  \text{where } R & \in \text{P}.
  \end{align*}

Counting problems

• examples
  – \#\text{SAT}: given 3-CNF \( \phi \) how many satisfying assignments are there?

  – \#\text{CLIQUE}: given \( (G, k) \) how many cliques of size at least \( k \) are there?
Reductions

- Reduction from function problem \( f_1 \) to function problem \( f_2 \)
  - two efficiently computable functions \( Q, A \)

\[
\begin{array}{c}
\text{(prob. 1)} \\
f_1(x)
\end{array}
\begin{array}{c}
Q
\end{array}
\begin{array}{c}
\text{(prob. 2)}
\end{array}
\begin{array}{c}
f_2(y)
\end{array}
\begin{array}{c}
f_2(f_1(x))
\end{array}
\begin{array}{c}
f_1(x)
\end{array}
\begin{array}{c}
A
\end{array}
\begin{array}{c}
f_2(y)
\end{array}
\]

- Problem \( f \) is \( \#P \)-complete if
  - \( f \) is in \( \#P \)
  - every problem in \( \#P \) reduces to \( f \)

- "parsimonious reduction": \( A \) is identity
  - many standard \( NP \)-completeness reductions are parsimonious
  - therefore: if \( \#SAT \) is \( \#P \)-complete we get lots of \( \#P \)-complete problems

\#SAT

\#SAT: given 3-CNF \( \varphi \) how many satisfying assignments are there?

**Theorem:** \#SAT is \#P-complete.

- Proof:
  - clearly in \( \#P \): \( \langle \varphi, A \rangle \in R \iff A \text{ satisfies } \varphi \)
  - take any \( f \in \#P \) defined by \( R \in P \)

\#SAT

\[ f(x) = \lvert \{ y : (x, y) \in R \} \rvert \]

1 iff \( (x, y) \in R \)

- add new variables \( z \), produce \( \varphi \) such that
  - \( \exists z \varphi(x, y, z) = 1 \iff C(x, y) = 1 \)
  - for \( (x, y) \) such that \( C(x, y) = 1 \) this \( z \) is **unique**
  - hardwire \( x \)
  - \# satisfying assignments = \( \lvert \{ y : (x, y) \in R \} \rvert \)

Relationship to other classes

- To compare to classes of decision problems, usually consider \( P^{\#P} \)
  - which is a decision class
- easy: \( NP, coNP \subseteq P^{\#P} \)
- easy: \( P^{\#P} \subseteq \text{PSPACE} \)

**Toda's Theorem** (homework): \( PH \subseteq P^{\#P} \).
Bipartite Matchings

- **Definition:**
  - \( G = (U, V, E) \) bipartite graph with \( |U| = |V| \)
  - a **perfect matching** in \( G \) is a subset \( M \subseteq E \) that touches every node, and no two edges in \( M \) share an endpoint

Bipartite Matchings

- **#MATCHING:** given a bipartite graph \( G = (U, V, E) \) how many perfect matchings does it have?

**Theorem:** #MATCHING is #P-complete.
- But... can find a perfect matching in polynomial time!
  - counting itself must be difficult

Cycle Covers

- **Claim:** 1-1 correspondence between cycle covers in \( G' \) and perfect matchings in \( G \)
  - #MATCHING and #CYCLE-COVER parsimoniously reducible to each other

**Theorem:** #CYCLE-COVER is #P-complete.
- implies #MATCHING is #P-complete
Cycle Cover is #P-complete

- clause gadget corresponding to \((A \lor B \lor C)\) has "xor" gadget between outer 3 edges and \(A, B, C\)

- xor gadget ensures that exactly one of two edges can be in cover

Cycle Cover is #P-complete

- Proof outline (reduce from #SAT)
  \(\neg x_1 \lor x_2 \lor \neg x_3 \land \neg x_2 \lor x_1 \lor \neg x_3 \land \ldots \land \neg x_3 \lor x_1 \lor \neg x_2\)

- N.B. must avoid reducing SAT to MATCHING!

Cycle Cover is #P-complete

- Introduce edge weights
  - cycle cover weight is product of weights of its edges

- "implement" xor gadget by
  - weight of cycle cover that "obeys" xor multiplied by 4 \((\text{mod } N)\)
  - weight of cycle cover that "violates" xor multiplied by \(N\)

- weighted xor gadget:
  - weight of cycle cover that "obeys" xor multiplied by 4 \((\text{mod } N)\)
  - weight of cycle cover that "violates" xor multiplied by \(N\)

- variable gadgets
- clause gadgets
- xor gadgets (exactly 1 of two edges is in cover)
Cycle Cover is \#P-complete

- Simulating positive edge weights
  - need to handle 2, 3, 4, 5, ..., N-1

\begin{align*}
  &\text{variable gadgets: } x_1, x_2, \ldots, x_n \\
  &\text{xor gadget (exactly 1 of two edges is in cover)} \\
  &\text{clause gadget (exactly 1 of clauses is satisfied)}
\end{align*}

\[ (-x_1 \lor x_2 \lor \ldots \lor x_n) \land (-x_2 \lor x_1) \land \ldots \land (x_n \lor -x_1) \]

- \( m = \# \text{ xor gadgets} \); \( n = \# \text{ variables} \); \( N > 4^m 2^n \)
- \( \# \text{ covers (mod } N) = (4^m) (\# \text{ sat. assignments}) \)