Computational Complexity Theory, Fall 2010	October 1

Lecture 12: Toda's Theorem. Parallel Computation and Circuits

Lecturer: Kristoffer Arnsfelt Hansen — Scribe: Mads Chr. Olesen

Recall: $\#P$:

- $f : \Sigma^* \to \mathbb{N}$
- There exists a polynomial p and $R \in P$
- $f(x) = |\{y \in \{0,1\}^{p(|x|)} | \langle x, y\rangle \in R\}|$

$P^{\#P}$ = class of languages computable in polynomial time with oracle access to $\#P$.
$\oplus P$ = is the number of witnesses odd

Theorem 1 (Toda, cont.)**.** $PH \subseteq P^{\#P}$

Proof. **First part:**

For any k there exists a randomized algorithm A that on input $(\psi, 1^m)$ where ψ is $\Sigma_k SAT$ instance, in polynomial time outputs a Boolean formula $A(\psi) = \phi$ s.t.

$$\psi \in \Sigma_k SAT \Rightarrow Pr[\phi \in \oplus SAT] \geq 1 - 2^{-m}$$

$$\psi \notin \Sigma_k SAT \Rightarrow Pr[\phi \in \oplus SAT] \geq 2^{-m}$$

$\#\phi$ = number of satisfying assignments to ϕ.

Lemma 2. *Given formulas ϕ and ψ we can in polynomial time compute "$\phi \cdot \psi$" and "$\phi + \psi$" such that*

$$\#(\phi \cdot \psi) = \#\phi \cdot \#\psi \qquad\qquad \#(\phi + \psi) = \#\phi + \#\psi$$

Modulus amplifying polynomials

Define $p(x) = 3x^4 + 4x^3$.

Lemma 3. *For any integers x and n:*

$$x \equiv -1 \pmod{n} \Rightarrow p(x) \equiv -1 \pmod{n^2}$$

$$x \equiv 0 \pmod{n} \Rightarrow p(x) \equiv 0 \pmod{n^2}$$

Proof. Assume $x \equiv -1 \pmod{n}$, that is, $n|(x+1)$. Then $n^2|(x+1)^2$, i.e., $n^2|x^2 + 2x + 1 \Leftrightarrow x^2 \equiv -2x - 1 \pmod{n^2}$. We can now evaluate $p(x)$ modulo n^2, substituting for x^2:

$$\begin{aligned} p(x) &\equiv 3x^4 + 4x^3 \equiv 3(2x-1)^2 - 4x(2x+1) \\ &\equiv 3(4x^2 + 4x + 1) - 8x^2 - 4x \\ &\equiv 4x^2 + 8x + 3 \\ &\equiv -4(2x+1) + 8x + 3 \equiv -1 \pmod{n^2} \end{aligned}$$

Assume $x \equiv 0 \pmod{n}$, that is $n|x$. Then $n^2|x^2 \Leftrightarrow x^2 \equiv 0 \pmod{n^2}$.
Thus $p(x) \equiv 3x^4 + 4x^3 \equiv 0 \pmod{n^2}$. $\square$

Corollary 4. *There is a polynomial Time algorithm B that on input $(\psi, 1^l)$ outputs a formula $B(\psi) = \phi$ such that*

$$\psi \in \oplus SAT \Rightarrow \#\phi \equiv -1 \pmod{2^{l+1}}$$

$$\psi \notin \oplus SAT \Rightarrow \#\phi \equiv 0 \pmod{2^{l+1}}$$

Proof. We can transfer ψ to

$$"3" \cdot (((\psi \cdot \psi) \cdot \psi) \cdot \psi) + "4" \cdot ((\psi \cdot \psi) \cdot \psi)$$

where "3" is a formula with 3 satisfying assignments, and "4" analogously.

Iterate this construction l times. □

Proof of Toda's Theorem:

Combine the algorithm A and B from above (for A we set $m = 2$):

$$\psi \in \Sigma_k SAT \Rightarrow Pr[\#B(A(\psi)) \equiv -1 \pmod{2^{l+1}}] \geq \frac{3}{4}$$

$$\psi \notin \Sigma_k SAT \Rightarrow Pr[\#B(A(\psi)) \equiv -1 \pmod{2^{l+1}}] \leq \frac{1}{4}$$

Suppose A uses R random bits. Let $A(\psi, r)$ denote the output of A when using $r \in \{0,1\}^R$ as the random bits.

Define the function $f : \Sigma^* \to \mathbb{N}$ s.t. $f(\psi) = \sum_{r \in \{0,1\}^R} \#B(A(\psi, r))$

Observe that f is a $\#P$ function: Witnesses are pairs (r, y) where $r \in \{0,1\}^R$ and y is assignment to $B(A(\psi, r))$.

Suppose $\psi \in \Sigma_k SAT$. Then $(f(\psi) \pmod{2^{l+1}}) \in [-2^R, -\frac{3}{4}2^R]$. This holds since $\#B(A(\psi, r)) \equiv -1$ for $\geq \frac{3}{4}2^R$ strings r, and 0 otherwise.

Suppose now $\psi \notin \Sigma_k SAT$: Then $(f(\psi) \pmod{2^{l+1}}) \in [-\frac{1}{4}2^R, 0]$. This holds since $\#B(A(\psi, r)) \equiv -1$ for $\leq \frac{1}{4}2^R$ strings r, and 0 otherwise.

For $l \geq R$ the intervals $[-2^R, -\frac{3}{4}2^R]$ and $[-\frac{1}{4}2^R, 0]$ do not overlap modulo 2^{l+1}, and we can thus decide whether $\psi \in \Sigma_k SAT$ from the value of $f(\psi)$. □

Counting hierarchy

We have thus proved that $P^{\#P}$ contains the entire polynomial hierarchy, showing the power of couting. Next step one could take would be to build a hierarchy based on counting analogously to the polynomial time hierarchy. This is the so-called counting hierarchy (which we will however not cover in any more detail).

$$P \subseteq NP \subseteq NP^{NP} \subseteq \ldots \subseteq PH \subseteq P^{\#P} \subseteq P^{\#P^{\#P}} \ldots \subseteq PSPACE$$

Polynomial hierarchy *Counting hierarchy*

Boolean Circuits

Recall $P \subseteq P/poly$. So if $NP \notin P/poly$ then $P \neq NP$. Hence one "approach" to separating P and NP would be to prove that a language in NP is not computing by polynomial size Boolean circuits. The Karp-Lipton theorem proved previosly shows that if something "a bit more" than $P \neq NP$ holds, namely that the polynomial time hierarchy does not collapse to the second level, then the approach is in fact has a possibility of success.

Theorem 5 (Karp-Lipton)**.** *If $NP \subseteq P/poly$ then $PH = \Sigma_2^P$, equivalently $PH \neq \Sigma_2^P \Rightarrow NP \not\subseteq P/poly$.*

Our next result will show that there are Boolean functions that require exponential sized Boolean circuits. (And in fact the vast majority of Boolean functions have this property)

Theorem 6 (Shannon)**.** *For any $n > 2$ there is a Boolean function $f : \{0,1\}^n \to \{0,1\}$ such that no Boolean circuit of size $\frac{2^n}{2n}$ computes f.*

Proof. By counting argument.

$$\#(\text{Boolean functions } f : \{0,1\}^n \to \{0,1\}) = 2^{2^n}$$

(We need to specify a Boolean value for each of the 2^n possible inputs.)

$$\#(\text{Boolean circuits on } n \text{ inputs with } S \text{ noninput gates}) \leq (3(n+S)^2)^S$$

(We just need to specify type (AND,OR,NOT) of each gate, and which input gates it takes).

Since each circuit can compute only one Boolean function we have

$$\#(\text{Boolean functions on } n \text{ inputs computed by a Boolean circuit of size } \frac{2^n}{2n}) \leq (3(n+\frac{2^n}{2n})^2)^{\frac{2^n}{2n}}$$

Compare with 2^{2^n}. Take log's:

$$\begin{aligned} 2^n \quad \text{vs.} \quad & \frac{2^n}{2n}(\log 3 + 2\log(n+\frac{2^n}{2n})) \\ & \leq \frac{2^n}{2n}(2 + 2\log(\frac{2^n}{n})) \\ & = \frac{2^n}{2n}(2 + 2(n - \log n)) \\ & = 2^n(1 - \frac{\log n}{n} + \frac{1}{n}) \end{aligned}$$

$\square$

Parallel Computation

Definition 7 (Efficient Parallel Computation)**.** On input length n:

- $n^{O(1)}$ many processors
- $\log^{O(1)} n$ time bound

Definition 8. Boolean circuits (general model)

- Directed acyclic graph
- Nodes are gates
- Gates are labelled with functions $g : \{0,1\}^k \to \{0,1\}$, k is the fanin

Examples:
$AND(y_1, \ldots, y_k) = y_1 \wedge \ldots \wedge y_k$
$OR(y_1, \ldots, y_k) = y_1 \vee \ldots \vee y_k$
$\oplus(y_1, \ldots, y_k) = y_1 \oplus \ldots \oplus y_k$

size = # wires in circuit ≈ #processors in parallel model
depth = length of longest path from input to output ≈ time in parallel model

Definition 9 (Nick's class)**.** NC = class of languages L where x_L can be computed by a family of fanin ≤ 2 circuits using AND, OR and NOT gates of $n^{O(1)}$ size and $\log^{O(1)} n$ depth.

NC hierarchy: $NC^1 \subseteq NC^2 \subseteq \ldots \subseteq NC^i$, where NC^i is the languages computed by $n^{O(1)}$ size, depth $O(\log^i(n))$, fanin ≤ 2 circuit families using AND, OR and NOT gates.

AC hierarchy: $AC^1 \subseteq AC^2 \subseteq \ldots \subseteq AC^i$, where AC^i is the languages computed by $n^{O(1)}$ size, depth $O(\log^i(n))$ (and arbitrary fanin) circuit families using AND, OR and NOT gates.

Proposition 10. *For all i, $AC^i \subseteq NC^{i+1} \subseteq AC^{i+1}$.*

Proof. Replace each AND and OR gate on k-inputs by binary tree of depth $O(\log k)$. Note $k = n^{O(1)}$ thus $O(\log k) = O(\log n)$. □