Computational Complexity Theory

Lecture 17: Switching lemma; PTMs
Class BPP

Department of Computer Science,
Indian Institute of Science

Recap: The Parity function

- PARITY$(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n$.
- Fact. PARITY$(x_1, x_2, \ldots, x_n)$ can be computed by a circuit of size $O(n)$ and a formula of size $O(n^2)$.
- Theorem. (*Khrapchenko 1971*) Any formula computing PARITY$(x_1, x_2, \ldots, x_n)$ has size $\Omega(n^2)$.
- Can poly-size constant depth circuits compute PARITY? No!

Recap: Depth 2 & 3 circuits for Parity

- Without loss of generality, a depth 2 circuit is either a DNF or a CNF.
- Obs. Any DNF computing PARITY has $\geq 2^{n-1}$ terms.
- Obs. There's a $2^{O(\sqrt{n})}$ size depth 3 circuit for PARITY.

Recap: Depth d circuit for Parity

- Obs. There's a $\exp(n^{1/(d-1)})$ size depth d circuit for PARITY, where $\exp(x) = 2^x$. *(Homework)*

- Is the $\exp(n^{1/(d-1)})$ upper bound on the size of depth d circuits computing PARITY tight? "Yes"

Recap: Lower bound for depth d circuits

- Theorem. *(Furst, Saxe, Sipser '81; Ajtai '83; Hastad '86)* Any depth d circuit computing PARITY has size $\exp(\Omega_d(n^{1/(d-1)}))$, where $\Omega_d()$ is hiding a d^{-1} factor.

- Furst, Saxe and Sipser showed a quasi-polynomial lower bound.
- Ajtai showed an exponential lower bound, but the bound wasn't optimal.
- Hastad showed an $\exp(\Omega(n^{1/(d-1)}))$ lower bound.
- *Rossman (2015)* showed an optimal $\exp(\Omega(dn^{1/(d-1)}))$ lower bound.

Recap: Lower bound for depth d circuits

- Theorem. *(Furst, Saxe, Sipser '81; Ajtai '83; Hastad '86)* Any depth d circuit computing PARITY has size $\exp(\Omega_d(n^{1/(d-1)}))$, where $\Omega_d()$ is hiding a d^{-1} factor.

- Gives a super-polynomial lower bound for depth d circuits for d up to o(log n).

- A lower bound for circuits of depth d = O(log n) implies a Boolean formula lower bound!

Recap: Lower bound for depth d circuits

- Theorem. *(Furst, Saxe, Sipser '81; Ajtai '83; Hastad '86)* Any depth d circuit computing PARITY has size $\exp(\Omega_d(n^{1/(d-1)}))$, where $\Omega_d()$ is hiding a d^{-1} factor.
- Proof idea. A ***random assignment*** to a "large" fraction of the variables makes a constant depth circuit of polynomial size evaluate to a constant (i.e., the circuit stops depending on the unset variables). On the other hand, we cannot make PARITY evaluate to a constant by setting less than n variables.

Recap: Lower bound for depth d circuits

- Theorem. *(Furst, Saxe, Sipser '81; Ajtai '83; Hastad '86)* Any depth d circuit computing PARITY has size $\exp(\Omega_d(n^{1/(d-1)}))$, where $\Omega_d()$ is hiding a d^{-1} factor.
- Proof idea. A ***random assignment*** to a "large" fraction of the variables makes a constant depth circuit of polynomial size evaluate to a constant (i.e., the circuit stops depending on the unset variables).

- We'll prove this fact using Hastad's **Switching lemma**. But first let us discuss some structural simplifications of depth d circuits.

Recap: Random restrictions

- A *restriction* σ is a partial assignment to a subset of the n variables.

- A *random restriction* σ that leaves m variables alive/unset is obtained by picking a random subset $S \subseteq [n]$ of size $n-m$ and setting every variable in S to 0/1 uniformly and independently.

- Let f_σ denote the function obtained by applying the restriction σ on f.

The Switching Lemma

- Switching lemma. Let f be a t-CNF on n variables and σ a random restriction that leaves $m = pn$ variables alive, where $p < 1/2$. Then,

  $\Pr_\sigma [f_\sigma$ can't be represented as a k-DNF$] \leq (16pt)^k$.

The Switching Lemma

- Switching lemma. Let f be a t-CNF on n variables and σ a random restriction that leaves $m = pn$ variables alive, where $p < 1/2$. Then,

  $\Pr_\sigma [f_\sigma$ can't be represented as a k-DNF$] \leq (16pt)^k$.

- We can interchange "CNF" and "DNF" in the above statement by applying the lemma on $\neg f$.

The Switching Lemma

- Switching lemma. Let f be a t-CNF on n variables and σ a random restriction that leaves $m = pn$ variables alive, where $p < 1/2$. Then,

  $\Pr_\sigma [f_\sigma$ can't be represented as a k-DNF$] \leq (16pt)^k$.

- We can interchange "CNF" and "DNF" in the above statement by applying the lemma on $\neg f$.

- We have used the lemma in the last lecture to prove the lower bound for AC^0 circuits computing parity.

Proof of the Switching Lemma

- Switching lemma. Let f be a t-CNF on n variables and σ a random restriction that leaves $m = pn$ variables alive, where $p < 1/2$. Then,

  $\Pr_\sigma [f_\sigma$ can't be represented as a k-DNF$] \leq (16pt)^k$.

- Proof. We'll present a proof due to Razborov.

Proof of the Switching Lemma

- Switching lemma. Let f be a t-CNF on n variables and σ a random restriction that leaves $m = pn$ variables alive, where $p < 1/2$. Then,

  $\Pr_\sigma [f_\sigma$ can't be represented as a k-DNF$] \leq (16pt)^k$.

- Proof. Let A_ℓ be the set of restrictions that keeps ℓ variables alive. Then, $|A_\ell| = \binom{n}{\ell}.2^{n-\ell}$.

Proof of the Switching Lemma

- Switching lemma. Let f be a t-CNF on n variables and σ a random restriction that leaves $m = pn$ variables alive, where $p < 1/2$. Then,

  $\Pr_\sigma [f_\sigma$ can't be represented as a k-DNF$] \leq (16pt)^k$.

- Proof. Let A_ℓ be the set of restrictions that keeps ℓ variables alive. Then, $|A_\ell| = \binom{n}{\ell}.2^{n-\ell}$. Let $B_{m,k} \subseteq A_m$ be the set of "bad" restrictions, i.e., a $\sigma \in A_m$ is in $B_{m,k}$ iff f_σ can't be represented as a k-DNF.
- We need to upper bound $|B_{m,k}|$.

Proof of the Switching Lemma

- Switching lemma. Let f be a t-CNF on n variables and σ a random restriction that leaves $m = pn$ variables alive, where $p < 1/2$. Then,

  $\Pr_\sigma [f_\sigma$ can't be represented as a k-DNF$] \leq (16pt)^k$.

- Proof. Let A_ℓ be the set of restrictions that keeps ℓ variables alive. Then, $|A_\ell| = \binom{n}{\ell}.2^{n-\ell}$. Let $B_{m,k} \subseteq A_m$ be the set of "bad" restrictions, i.e., a $\sigma \in A_m$ is in $B_{m,k}$ iff f_σ can't be represented as a k-DNF.
- We need to upper bound $|B_{m,k}|$.
- This is done by giving an ***injective map*** from $B_{m,k}$ to $A_{m-k} \times U$, where $U = \{0,1\}^{k(\log t + 2)}$. $|U| = (4t)^k$.

Proof of the Switching Lemma

- Switching lemma. Let f be a t-CNF on n variables and σ a random restriction that leaves $m = pn$ variables alive, where $p < 1/2$. Then,

  $\Pr_\sigma [f_\sigma$ can't be represented as a k-DNF$] \leq (16pt)^k$.

- Proof. Then, $|B_{m,k}| \leq \binom{n}{m-k} . 2^{n-m+k} . (4t)^k$. and so

  $|B_{m,k}|/|A_m| \leq [(m! \,.\, (n-m)!) / ((m-k)! \,.\, (n-m+k)!)] . 2^k . (4t)^k$

Proof of the Switching Lemma

- Switching lemma. Let f be a t-CNF on n variables and σ a random restriction that leaves $m = pn$ variables alive, where $p < 1/2$. Then,

  $\Pr_\sigma [f_\sigma$ can't be represented as a k-DNF$] \leq (16pt)^k$.

- Proof. Then, $|B_{m,k}| \leq \binom{n}{m-k} . 2^{n-m+k} . (4t)^k$. and so

$$|B_{m,k}|/|A_m| \leq [(m! \,.\, (n-m)!) \,/\, ((m-k)! \,.\, (n-m+k)!)] . 2^k . (4t)^k$$
$$\leq (m/(n-m))^k \,.\, 2^k \,.\, (4t)^k$$
$$= (p/(1-p))^k \,.\, 2^k \,.\, (4t)^k \qquad \text{(as } m = pn\text{)}$$
$$\leq p^k \,.\, 2^k \,.\, 2^k \,.\, (4t)^k \qquad \text{(as } p < 1/2 \text{)}$$
$$= (16pt)^k \,.$$

Proof of the Switching Lemma

- Switching lemma. Let f be a t-CNF on n variables and σ a random restriction that leaves $m = pn$ variables alive, where $p < 1/2$. Then,

  $\Pr_\sigma [f_\sigma$ can't be represented as a k-DNF$] \leq (16pt)^k$.

- Proof. Next, we show an injection from $B_{m,k}$ to $A_{m-k} \times U$, where $U = \{0,1\}^{k(\log t + 2)}$.

A definition and a notation

- Definition. A *min-term* of a function g is a restriction π such that $g_\pi = 1$, but **no** proper sub-restriction of π makes g evaluate to 1.
- Obs. If g can't be expressed as a k-DNF, then g has a min-term π of *width* $> k$ (i.e., π assigns 0/1 values to more than k variables). (*Homework*)

A definition and a notation

- Definition. A *min-term* of a function g is a restriction π such that $g_\pi = 1$, but no proper sub-restriction of π makes g evaluate to 1.
- Obs. If g can't be expressed as a k-DNF, then g has a min-term π of width $> k$ (i.e., π assigns 0/1 values to more than k variables). (*Homework*)
- Notation. If σ is a restriction that assigns 0/1 values to variables in $S_1 \subseteq [n]$ and π is a restriction that assigns 0/1 values to variables in $S_2 \subseteq [n] \setminus S_1$, then $\sigma \circ \pi$ is the "composed" restriction that assigns 0/1 values to $S_1 \cup S_2$ consistent with σ and π. $|\pi|$:= width of π.

Injection from $B_{m,k}$ to $A_{m-k} \times U$

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width $> k$.

Injection from $B_{m,k}$ to A_{m-k} x U

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width > k.
- A map χ from $B_{m,k}$ to A_{m-k} x U : (*Overview*)

➢ **Step 1:** For $\sigma \in B_{m,k}$, let π be the lexicographically smallest min-term of f_σ of width > k. We'll carefully define a sub-restriction π' of π of width k.

Injection from $B_{m,k}$ to A_{m-k} x U

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width > k.
- A map χ from $B_{m,k}$ to A_{m-k} x U : (*Overview*)

➢ **Step 1:** For $\sigma \in B_{m,k}$, let π be the lexicographically smallest min-term of f_σ of width > k. We'll carefully define a sub-restriction π' of π of width k.

➢ **Step 2:** Using π', we'll carefully define a restriction ρ that assigns 0/1 values to the same set of variables as π'.

Injection from $B_{m,k}$ to A_{m-k} x U

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width > k.
- A map χ from $B_{m,k}$ to A_{m-k} x U : (*Overview*)

➢ **Step 1:** For $\sigma \in B_{m,k}$, let π be the lexicographically smallest min-term of f_σ of width > k. We'll carefully define a sub-restriction π' of π of width k.

➢ **Step 2:** Using π', we'll carefully define a restriction ρ that assigns 0/1 values to the same set of variables as π'.

➢ **Step 3:** Using π', define a $u \in U$. Finally, $\chi(\sigma) := (\sigma \circ \rho, u)$.

Injection from $B_{m,k}$ to A_{m-k} x U

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width > k.
- A map χ from $B_{m,k}$ to A_{m-k} x U :

➢ **Step 1:** For $\sigma \in B_{m,k}$, let π be the lexicographically smallest min-term of f_σ of width > k. Order the clauses of f, and order the ≤ t variables appearing within such a clause.

Injection from $B_{m,k}$ to A_{m-k} x U

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width > k.
- A map χ from $B_{m,k}$ to A_{m-k} x U :

➢ **Step 1:** For $\sigma \in B_{m,k}$, let π be the lexicographically smallest min-term of f_σ of width > k. Order the clauses of f, and order the ≤ t variables appearing within such a clause. C_1 be the first *surviving* clause in f_σ and $\pi(1)$ the assignment to its surviving variables made by π.

Injection from $B_{m,k}$ to A_{m-k} x U

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width > k.
- A map χ from $B_{m,k}$ to A_{m-k} x U :

➢ **Step 1:** For $\sigma \in B_{m,k}$, let π be the lexicographically smallest min-term of f_σ of width > k. Order the clauses of f, and order the ≤ t variables appearing within such a clause. C_1 be the first surviving clause in f_σ and $\pi(1)$ the assignment to its surviving variables made by π. C_2 be the first surviving clause in $f_{\sigma\circ\pi(1)}$ and $\pi(2)$ the assignment to its surviving variables made by π.

Injection from $B_{m,k}$ to $A_{m-k} \times U$

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width $> k$.
- A map χ from $B_{m,k}$ to $A_{m-k} \times U$:

➢ **Step 1:** For $\sigma \in B_{m,k}$, let π be the lexicographically smallest min-term of f_σ of width $> k$. Order the clauses of f, and order the $\leq t$ variables appearing within such a clause. C_1 be the first surviving clause in f_σ and $\pi(1)$ the assignment to its surviving variables made by π. C_2 be the first surviving clause in $f_{\sigma \circ \pi(1)}$ and $\pi(2)$ the assignment to its surviving variables made by π. Continue like this.. Stop if $|\pi(1) \circ \ldots \circ \pi(r)| \geq k$.

Injection from $B_{m,k}$ to A_{m-k} x U

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width > k.
- A map χ from $B_{m,k}$ to A_{m-k} x U :

➢ **Step 1:** If $|\pi(1)\circ\ldots\circ\pi(r)| > k$, then "prune" $\pi(r)$ by restricting it to the set of "smallest" variables in C_r so that $|\pi(1)\circ\ldots\circ\pi(r)| = k$. Define $\pi' := \pi(1)\circ\ldots\circ\pi(r)$; $|\pi'| = k$.

Injection from $B_{m,k}$ to A_{m-k} x U

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width > k.
- A map χ from $B_{m,k}$ to A_{m-k} x U :

➢ **Step 2:** For $i \in [r]$, let S_i be the set of variables in the clause C_i that are assigned 0/1 values by $\pi(i)$. $|S_i| = |\pi(i)|$. Let $\rho(i)$ be the *unique* assignment to the variables in S_i that makes the corresponding literals in C_i zero. Define $\rho := \rho(1) \circ \ldots \circ \rho(r)$.

Injection from $B_{m,k}$ to A_{m-k} x U

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width > k.
- A map χ from $B_{m,k}$ to A_{m-k} x U :

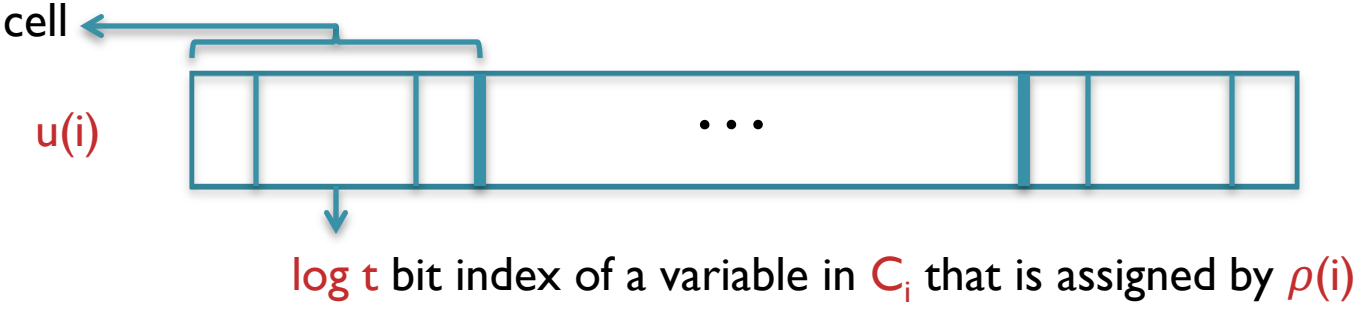
➢ **Step 2:** For $i \in [r]$, let S_i be the set of variables in the clause C_i that are assigned 0/1 values by $\pi(i)$. $|S_i| = |\pi(i)|$. Let $\rho(i)$ be the *unique* assignment to the variables in S_i that makes the corresponding literals in C_i zero. Define $\rho := \rho(1) \circ \ldots \circ \rho(r)$.

➢ Remark*. $\pi(i)$ and $\rho(i)$ are assignments to the same set of variables S_i. C_i remains unsatisfied under $\rho(i)$.

Injection from $B_{m,k}$ to A_{m-k} x U

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width > k.
- A map χ from $B_{m,k}$ to A_{m-k} x U :

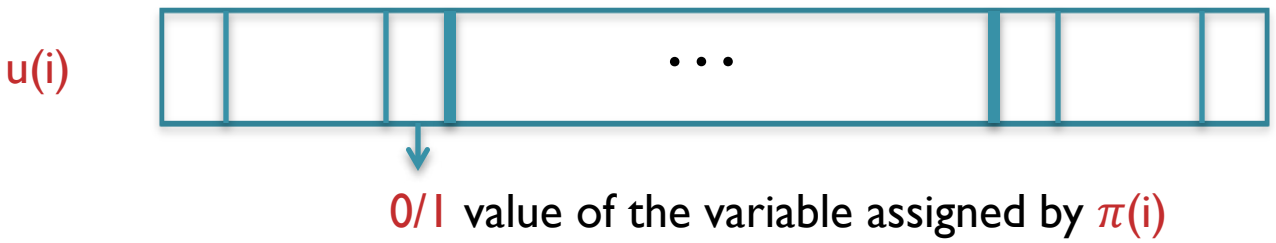
➢ **Step 3:** For $i \in [r]$, let u(i) be the string obtained by listing the indices (*within* the clause C_i) of the variables assigned by $\rho(i)$ along with the values assigned to them by $\pi(i)$.



Injection from $B_{m,k}$ to A_{m-k} x U

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width > k.
- A map χ from $B_{m,k}$ to A_{m-k} x U :

➢ **Step 3:** For $i \in [r]$, let u(i) be the string obtained by listing the indices (*within* the clause C_i) of the variables assigned by $\rho(i)$ along with the values assigned to them by $\pi(i)$.



Injection from $B_{m,k}$ to A_{m-k} x U

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width > k.
- A map χ from $B_{m,k}$ to A_{m-k} x U :

➢ **Step 3:** For i∈[r], let u(i) be the string obtained by listing the indices (*within* the clause C_i) of the variables assigned by $\rho(i)$ along with the values assigned to them by $\pi(i)$.

Injection from $B_{m,k}$ to A_{m-k} x U

- f is a t-CNF on n variables. $U = \{0,1\}^{k(\log t + 2)}$.
- A_ℓ = set of restrictions that keeps ℓ variables alive.
- $B_{m,k} = \{\sigma \in A_m : f_\sigma$ can't be represented as a k-DNF$\}$.
- Obs. If $\sigma \in B_{m,k}$ then f_σ has a min-term of width > k.
- A map χ from $B_{m,k}$ to A_{m-k} x U :

➢ **Step 3:** For $i \in [r]$, let u(i) be the string obtained by listing the indices (*within* the clause C_i) of the variables assigned by $\rho(i)$ along with the values assigned to them by $\pi(i)$. Define u by concatenating u(1), …, u(r) in order. Observe that |u| = k(log t + 2). Finally, $\chi(\sigma) := (\sigma \circ \rho\ ,\ u)$. (Remark. The delimiter bits make it possible to extract u(i) from u.)

Injection from $B_{m,k}$ to A_{m-k} x U

- We'll now show that it is possible to recover σ from $(\sigma \circ \rho\ , u)$ which will then imply χ is an injection.

Injection from $B_{m,k}$ to $A_{m-k} \times U$

- We'll now show that it is possible to recover σ from $(\sigma \circ \rho\ , u)$ which will then imply χ is an injection.
- Obs*. For every $i \in [r]$, the first "unsatisfied" clause in $f_{\sigma \circ \pi(1) \circ \ldots \circ \pi(i-1) \circ \rho(i) \circ \ldots \circ \rho(r)}$ is C_i .
- Proof. Fix an $i \in [r]$. By construction, C_i is the first surviving clause in $f_{\sigma \circ \pi(1) \circ \ldots \circ \pi(i-1)}$.

Injection from $B_{m,k}$ to $A_{m-k} \times U$

- We'll now show that it is possible to recover σ from $(\sigma \circ \rho\ , u)$ which will then imply χ is an injection.
- Obs*. For every $i \in [r]$, the first "unsatisfied" clause in $f_{\sigma \circ \pi(1) \circ \ldots \circ \pi(i-1) \circ \rho(i) \circ \ldots \circ \rho(r)}$ is C_i .
- Proof. Fix an $i \in [r]$. By construction, C_i is the first surviving clause in $f_{\sigma \circ \pi(1) \circ \ldots \circ \pi(i-1)}$. C_i remains unsatisfied under $\rho(i)$ (Remark*). Further, $\rho(i+1), \ldots, \rho(r)$ do not touch any variable of C_i. Hence, C_i is the first unsatisfied clause in $f_{\sigma \circ \pi(1) \circ \ldots \circ \pi(i-1) \circ \rho(i) \circ \ldots \circ \rho(r)}$.

Injection from $B_{m,k}$ to A_{m-k} x U

- We'll now show that it is possible to recover σ from $(\sigma \circ \rho\ , u)$ which will then imply χ is an injection.
- Obs*. For every $i \in [r]$, the first "unsatisfied" clause in $f_{\sigma \circ \pi(1) \circ \ldots \circ \pi(i-1) \circ \rho(i) \circ \ldots \circ \rho(r)}$ is C_i .
- Recovering σ from $(\sigma \circ \rho\ , u)$:

➢ Pick the first unsatisfied clause in $f_{\sigma \circ \rho(1) \circ \ldots \circ \rho(r)}$. This clause is C_1 (Obs*). Now by looking at $u(1)$, we can derive $\pi(1)$.

Injection from $B_{m,k}$ to $A_{m-k} \times U$

- We'll now show that it is possible to recover σ from $(\sigma \circ \rho\ , u)$ which will then imply χ is an injection.
- Obs*. For every $i \in [r]$, the first "unsatisfied" clause in $f_{\sigma \circ \pi(1) \circ \ldots \circ \pi(i-1) \circ \rho(i) \circ \ldots \circ \rho(r)}$ is C_i .
- Recovering σ from $(\sigma \circ \rho\ , u)$:

➢ Pick the first unsatisfied clause in $f_{\sigma \circ \rho(1) \circ \ldots \circ \rho(r)}$. This clause is C_1 (Obs*). Now by looking at $u(1)$, we can derive $\pi(1)$. Construct $\sigma \circ \pi(1) \circ \rho(2) \circ \ldots \circ \rho(r)$ from $\sigma \circ \rho(1) \circ \ldots \circ \rho(r)$ and $\pi(1)$.

Injection from $B_{m,k}$ to $A_{m-k} \times U$

- We'll now show that it is possible to recover σ from $(\sigma \circ \rho\ , u)$ which will then imply χ is an injection.
- Obs*. For every $i \in [r]$, the first "unsatisfied" clause in $f_{\sigma \circ \pi(1) \circ \ldots \circ \pi(i-1) \circ \rho(i) \circ \ldots \circ \rho(r)}$ is C_i .
- Recovering σ from $(\sigma \circ \rho\ , u)$:

➢ Pick the first unsatisfied clause in $f_{\sigma \circ \pi(1) \circ \rho(2) \circ \ldots \circ \rho(r)}$. This clause is C_2 (Obs*). Now by looking at $u(2)$, we can derive $\pi(2)$. Construct $\sigma \circ \pi(1) \circ \pi(2) \circ \rho(3) \circ \ldots \circ \rho(r)$ from $\sigma \circ \pi(1) \circ \rho(2) \circ \ldots \circ \rho(r)$ and $\pi(2)$.

Injection from $B_{m,k}$ to A_{m-k} x U

- We'll now show that it is possible to recover σ from $(\sigma \circ \rho\ , u)$ which will then imply χ is an injection.
- Obs*. For every $i \in [r]$, the first "unsatisfied" clause in $f_{\sigma \circ \pi(1) \circ \ldots \circ \pi(i-1) \circ \rho(i) \circ \ldots \circ \rho(r)}$ is C_i .
- Recovering σ from $(\sigma \circ \rho\ , u)$:

➢ Continuing like this we can construct $\sigma \circ \pi(1) \circ \ldots \circ \pi(r)$ and also find $\pi(1), \ldots, \pi(r)$ in the process. From here, recovering σ is straightforward.

- Ref.
  https://sites.math.rutgers.edu/~sk1233/courses/topics-S13/lec3.pdf

Probabilistic Turing Machines

Randomized computation

- So far, we have used deterministic TMs to model "real-world" computation. But, DTMs don't have the ability to make *random choices* during a computation.

- The usefulness of randomness in computation was realized as early as the 1940s when the first electronic computer, ENIAC, was developed.

Randomized computation

- So far, we have used deterministic TMs to model "real-world" computation. But, DTMs don't have the ability to make *random choices* during a computation.

- The usefulness of randomness in computation was realized as early as the 1940s when the first electronic computer, ENIAC, was developed.
  - The use of statistical methods in a computational model of a thermonuclear reaction for the ENIAC led to the invention of the ***Monte Carlo methods***.

Randomized computation

- So far, we have used deterministic TMs to model "real-world" computation. But, DTMs don't have the ability to make *random choices* during a computation.

- The usefulness of randomness in computation was realized as early as the 1940s when the first electronic computer, ENIAC, was developed.

- To study randomized computation, we need to give TMs the power of generating random numbers.

Randomized computation

- How realistic such a randomized TM model would be depends on our ability to generate bits that are "close" to being truly random.

1 with probability ½
0 with probability ½

Randomized computation

- How realistic such a randomized TM model would be depends on our ability to generate bits that are "close" to being truly random.
- Many programming languages have built-in random number generator functions.
- Examples of pseudo-random number generators are *linear congruential generators* and von Neumann's *middle-square method*.

Randomized computation

- How realistic such a randomized TM model would be depends on our ability to generate bits that are "close" to being truly random.
- Many programming languages have built-in random number generator functions.
- Examples of pseudo-random number generators are *linear congruential generators* and von Neumann's *middle-square method*.

$$X_{i+1} = aX_i + c \pmod{m}$$

Randomized computation

- How realistic such a randomized TM model would be depends on our ability to generate bits that are "close" to being truly random.
- Many programming languages have built-in random number generator functions.
- Examples of pseudo-random number generators are *linear congruential generators* and von Neumann's *middle-square method*.

Square an n bit number to get a 2n bit number and take the middle n bits.

Randomized computation

- How realistic such a randomized TM model would be depends on our ability to generate bits that are "close" to being truly random.
- Many programming languages have built-in random number generator functions.
- Examples of pseudo-random number generators are *linear congruential generators* and von Neumann's *middle-square method*.
- To what extent a PRG is adequate is studied under the topic `*Pseudorandomness*' in complexity theory.

Randomized computation

- How realistic such a randomized TM model would be depends on our ability to generate bits that are "close" to being truly random.
- Many programming languages have built-in random number generator functions.
- Examples of pseudo-random number generators are *linear congruential generators* and von Neumann's *middle-square method*.
- We'll assume that a TM can generate, or has access to, truly random bits/coins. (We'll touch upon "truly vs biased random bits" at end of the lecture.)

Probabilistic Turing Machines

- Definition. A *probabilistic Turing machine* (PTM) M has two transition functions δ_0 and δ_1. At each step of computation on input $x \in \{0,1\}^*$, M applies one of δ_0 and δ_1 uniformly at random (independent of the previous steps). M outputs either 1 (accept) or 0 (reject).

Probabilistic Turing Machines

- Definition. A *probabilistic Turing machine* (PTM) M has two transition functions δ_0 and δ_1. At each step of computation on input $x \in \{0,1\}^*$, M applies one of δ_0 and δ_1 uniformly at random (independent of the previous steps). M outputs either 1 (accept) or 0 (reject). M runs in T(n) time if M always halts within T(|x|) steps *regardless of its random choices*.

Probabilistic Turing Machines

- Definition. A *probabilistic Turing machine* (PTM) M has two transition functions δ_0 and δ_1. At each step of computation on input $x \in \{0,1\}^*$, M applies one of δ_0 and δ_1 uniformly at random (independent of the previous steps). M outputs either 1 (accept) or 0 (reject). M runs in T(n) time if M always halts within T(|x|) steps *regardless of its random choices*.

- Note. PTMs and NTMs are syntatically similar – both have two transition functions.

Probabilistic Turing Machines

- Definition. A *probabilistic Turing machine* (PTM) M has two transition functions δ_0 and δ_1. At each step of computation on input $x \in \{0,1\}^*$, M applies one of δ_0 and δ_1 uniformly at random (independent of the previous steps). M outputs either 1 (accept) or 0 (reject). M runs in T(n) time if M always halts within T(|x|) steps *regardless of its random choices*.

- Note. But, semantically, they are quite different – unlike NTMs, PTMs are meant to model realistic computation devices.

Probabilistic Turing Machines

- Definition. A *probabilistic Turing machine* (PTM) M has two transition functions δ_0 and δ_1. At each step of computation on input $x \in \{0,1\}^*$, M applies one of δ_0 and δ_1 uniformly at random (independent of the previous steps). M outputs either 1 (accept) or 0 (reject). M runs in T(n) time if M always halts within T(|x|) steps *regardless of its random choices*.

- Note. The above definition allows a PTM M to not halt on some computation paths defined by its random choices (unless we explicitly say that M runs in T(n) time). More on this later when we define ZPP.

Class BPP

- Definition. A PTM M *decides* a language L in time T(n) if M runs in T(n) time, and for every $x \in \{0,1\}^*$,

$$\Pr[M(x) = L(x)] \geq 2/3.$$

- Definition. A language L is in BPTIME(T(n)) if there's PTM that decides L in O(T(n)) time.

Class BPP

- Definition. A PTM M *decides* a language L in time T(n) if M runs in T(n) time, and for every $x \in \{0,1\}^*$,

$$\Pr[M(x) = L(x)] \geq 2/3.$$

- Definition. A language L is in BPTIME(T(n)) if there's PTM that decides L in O(T(n)) time.

- Definition. $BPP = \bigcup_{c > 0} BPTIME(n^c)$.

- Clearly, $P \subseteq BPP$.

Class BPP

- Definition. A PTM M *decides* a language L in time T(n) if M runs in T(n) time, and for every $x \in \{0,1\}^*$,

$$\underbrace{\Pr[M(x) = L(x)]}_{\text{Success probability}} \geq 2/3.$$

- Definition. A language L is in BPTIME(T(n)) if there's PTM that decides L in O(T(n)) time.

- Definition. $BPP = \bigcup_{c > 0} BPTIME(n^c)$.

- Clearly, $P \subseteq BPP$.

Class BPP

- Definition. A PTM M *decides* a language L in time T(n) if M runs in T(n) time, and for every $x \in \{0,1\}^*$,

$$\Pr[M(x) = L(x)] \geq 2/3.$$

- Definition. A language L is in BPTIME(T(n)) if there's PTM that decides L in O(T(n)) time.

- Definition. $BPP = \bigcup_{c > 0} BPTIME(n^c)$.

- Clearly, $P \subseteq BPP$.

Remark. The defn of class BPP is robust. The class remains unaltered if we replace 2/3 by any constant ***strictly greater*** than (i.e., ***bounded away*** from) ½. We'll discuss this next.

Class BPP

- Definition. A PTM M *decides* a language L in time T(n) if M runs in T(n) time, and for every $x \in \{0,1\}^*$,

$$\Pr[M(x) = L(x)] \geq 2/3.$$

- Definition. A language L is in BPTIME(T(n)) if there's PTM that decides L in O(T(n)) time.

- Definition. $BPP = \bigcup_{c > 0} BPTIME(n^c)$.

Bounded-error **P**robabilistic **P**olynomial-time

- Clearly, $P \subseteq BPP$.

Remark. The defn of class BPP is robust. The class remains unaltered if we replace 2/3 by any constant ***strictly greater*** than (i.e., ***bounded away*** from) ½. We'll discuss this next.

Class BPP

- Definition. A PTM M *decides* a language L in time T(n) if M runs in T(n) time, and for every $x \in \{0,1\}^*$,

$$\Pr[M(x) = L(x)] \geq 2/3.$$

- Definition. A language L is in BPTIME(T(n)) if there's PTM that decides L in O(T(n)) time.

- Definition. $BPP = \bigcup_{c > 0} BPTIME(n^c)$.

- Clearly, $P \subseteq BPP$.

Remark. Achieving success probability ½ is trivial for any language. If we replace ≥ 2/3 by > ½ then the corresponding class is called PP, which is (presumably) larger than BPP. More on PP later.