18.404/6.840 Lecture 24

Last time:

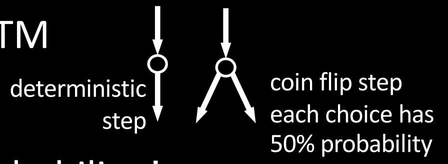
- Probabilistic computation
- The class BPP
- Branching programs
- Arithmetization
- Started showing $!"_{\text{ROBP}} \in \text{BPP}$

Today: (Sipser §10.2)

- Finish $!"_{\text{ROBP}} \in \text{BPP}$

Review: Probabilistic TMs and BPP

Defn: A probabilistic Turing machine (PTM) is a variant of a NTM where each computation step has 1 or 2 possible choices.

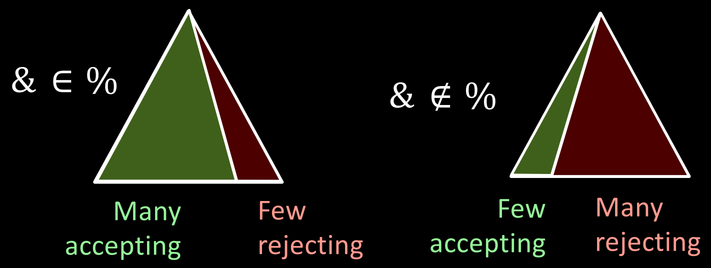


Defn: For ! ≥ 0 say PTM \$ <u>decides language %with error probability !</u> if for every &, Pr[\$ gives the wrong answer about & ∈ %] ≤ ! .

Defn: BPP = {%| some poly-time PTM decides %with error ! = +/ , }

⇩

Amplification lemma: $2^{-. /01(2)}$



Check-in 24.1

Actually using a probabilistic algorithm presupposes a source of randomness. Can we use a standard pseudo-random number generator (PRG) as the source?

(a) Yes, but the result isn't guaranteed.

(b) Yes, but it will run in exponential time.

(c) No, a TM cannot implement a PRG.

(d) No, because that would show P = BPP.

Check-in 24.1

Review: Branching Programs

Defn: A <u>branching program</u> (BP) is a directed, acyclic (no cycles) graph that has

1. *Query nodes* labeled x_i and having two outgoing edges labeled 0 and 1.
2. *Two output nodes* labeled 0 and 1 and having no outgoing edges.
3. A designated *start node*.

Theorem: EQ_{BP} is coNP-complete (on pset 6)

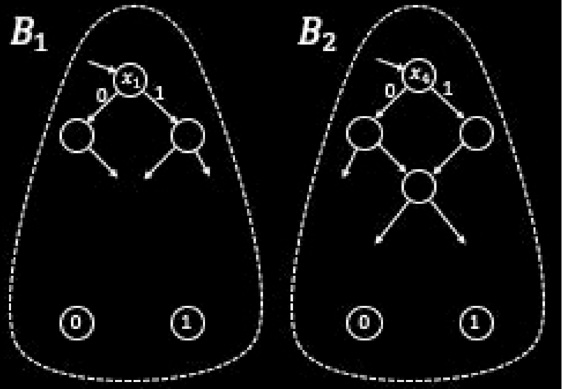
Defn: A BP is <u>read-once</u> if it never queries a variable more than once on any path from the start node to an output.

Defn: $EQ_{\text{ROBP}} = \{\langle B_1, B_2\rangle \mid B_1$ and B_2 are equivalent read-once BPs$\}$

Theorem: $EQ_{\text{ROBP}} \in$ BPP

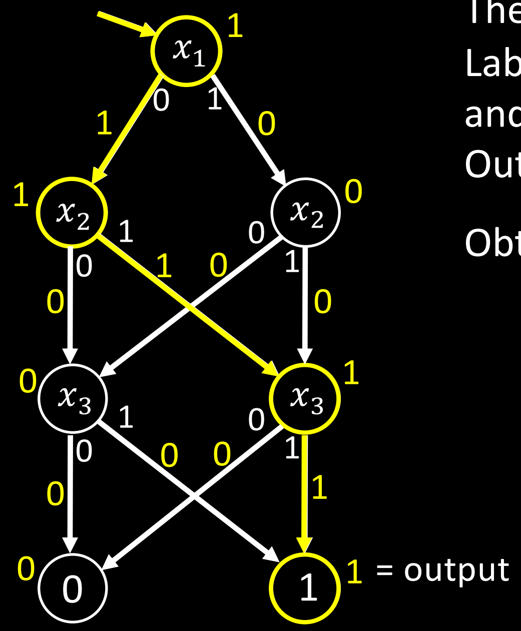
Proof idea: Run B_1 and B_2 on a randomly selected <u>non-Boolean input</u> and accept if get same output.

Method: Use <u>arithmetization</u> (simulating $\wedge$ and $\vee$ with $+$ and $\times$) to define BP operation on non-Boolean inputs.



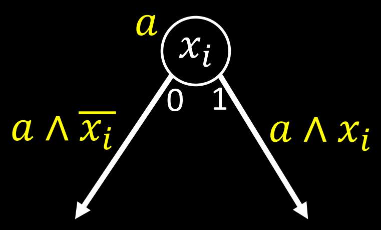
Boolean Labeling

Alternative way to view BP computation

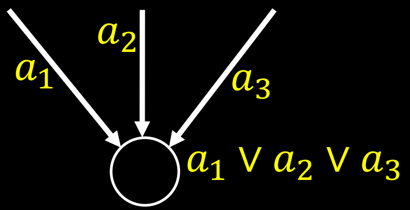


Show by example: Input is $x_1 = 0,\ x_2 = 1,\ x_3 = 1$
The BP follows its execution path.
Label all nodes and edges on the execution path with 1
and off the execution path with 0.
Output the label of the output node 1.

Obtain the labeling inductively by using these rules:



Label outgoing edges from nodes



Label nodes from incoming edges

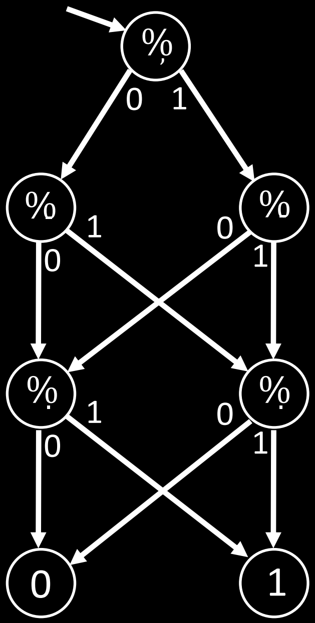
Arithmetization Method

Method: Simulate ∧ and ∨ with + and ×.

$$' \wedge / \;\to\; ' \times / = '/$$

$$\neg \;\to\; (1 - ')$$

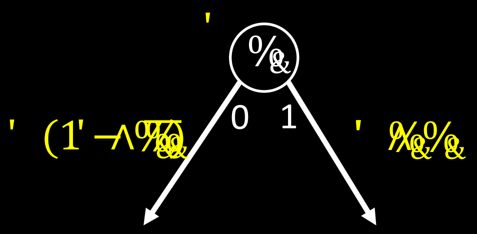
$$' \vee / \;\to\; ' + / - '/$$

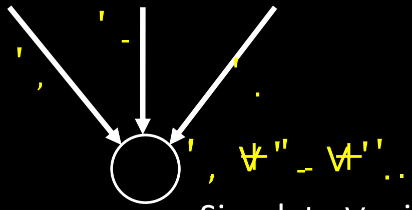


Replace Boolean labeling with arithmetical labeling

Inductive rules:

Start node labeled 1



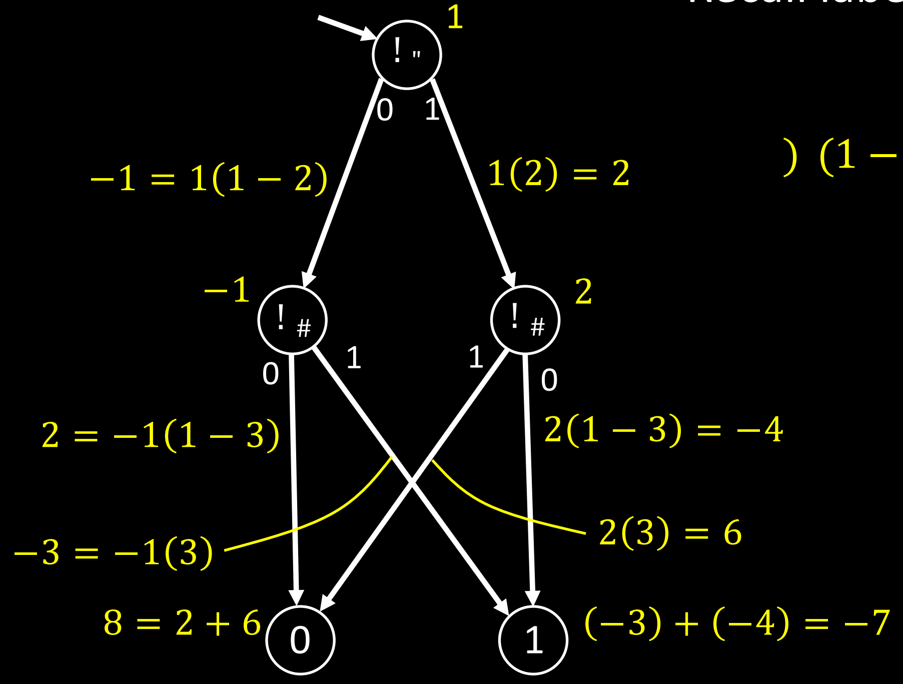


Simulate ∨ with + because the BP is acyclic.
The execution path can enter a node
at most one time.

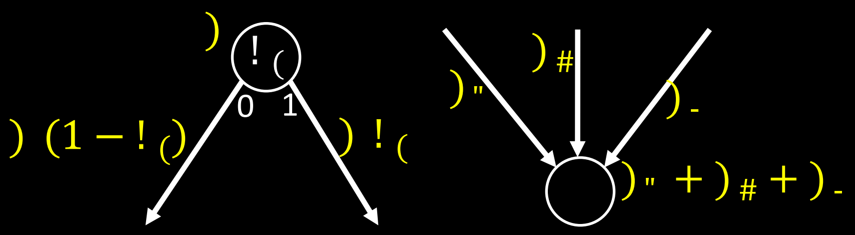
Non-Boolean Labeling

Use the arithmetized interpretation of the BP's computation to define its operation on non-Boolean inputs.

Example: $!_{"} = 2$, $!_{\#} = 3$



Recall labeling rules:



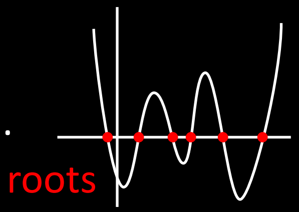
Algorithm sketch for 45_{ROBP}: "On input $\langle :_{"}, :_{\#} \rangle$

1. Pick a random *non-Boolean* input assignment.
2. Evaluate $:_{"}$ and $:_{\#}$ on that assignment.
3. If $:_{"}$ and $:_{\#}$ disagree then *reject*.
   If they agree then *accept*."

More details and correctness proof to come.
First some algebra…

Roots of Polynomials

Let $!(") = \$_{\%}"^{\&} + \$_{(}"^{\&)\,(} + \$_{*}"^{\&)\,*} + \cdots + \$_{\&}$ be a polynomial.
If , is some constant and $!(,) = 0$ call , a <u>root</u> of ! .



Polynomial Lemma: If $!(") \neq 0$ is polynomial of degree ≤ 0 then ! has ≤ 0 roots.
Proof by induction (see text).

Corollary 1: If $!_{(}(")$ and $!_{*}(")$ are both degree ≤ 0 and $!_{(} \neq !_{*}$
then $!_{(}(,) = !_{*}(,)$ for ≤ 0 values , .
Proof: Let $! = !_{(} - !_{*}$.

Above holds for any field 4 (a <u>field</u> is a set with $+$ and $\times$ operations that have typical properties).
We will use a finite field 4_{6} with 7 elements where 7 is prime and $+, \times$ operate mod 7.

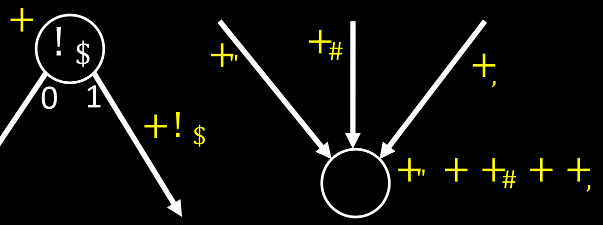
Corollary 2: If $!(") \neq 0$ has degree ≤ 0 and we pick a random $8 \in 4_{6}$, then $\Pr[\,!(8) = 0\,] \leq {}^{\&}/_{6}$.
Proof: There are at most 0 roots out of 7 possibilities.

Theorem (Schwartz-Zippel): If $!("_{(}, \ldots, "_{=}) \neq 0$ has degree ≤ 0 in each $"_{>}$ and
we pick random $8_{(}, \ldots, 8_{=} \in 4_{6}$ then $\Pr[\,!(8_{(}, \ldots, 8_{=}) = 0\,] \leq {}^{=\&}/_{6}$
Proof by induction (see text).

Symbolic Execution

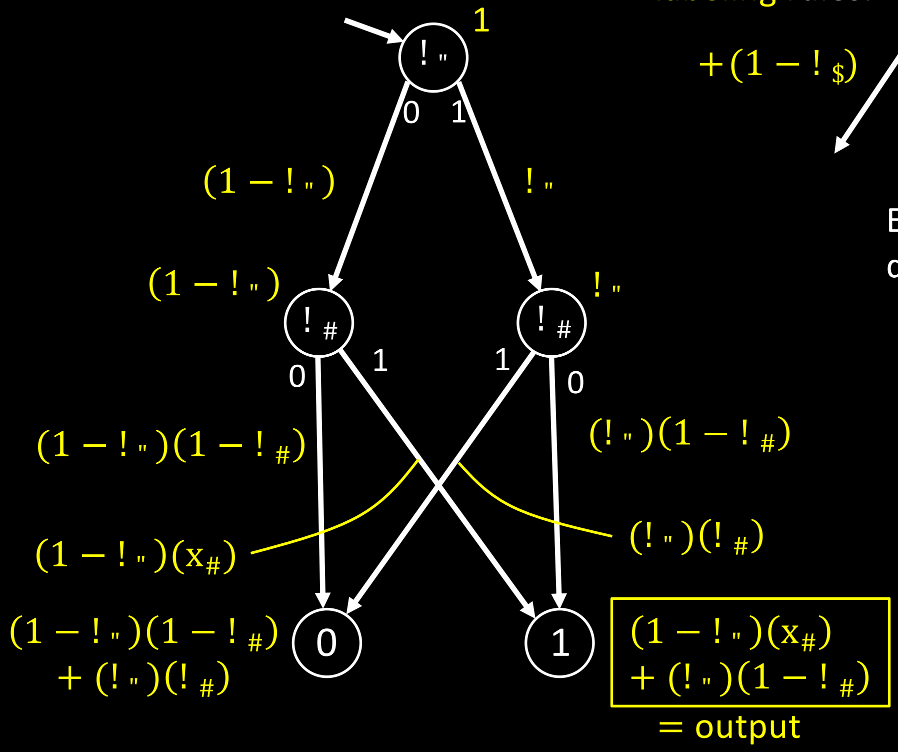
Leave the $!_{\$}$ as variables and obtain an expression in the $!_{\$}$ for the output of the BP.

Recall labeling rules:



Exponents ≤ 1 due to “read-once”

Assume read exactly once so that for each 3 $(!_{\$})$ or $(1-!_{\$})$ appears in every row



$$\boxed{\begin{gathered}(1-!_{"})(x_{\#})\\ +(!_{"})(1-!_{\#})\end{gathered}}$$

$=$ output

$$\boxed{\text{form of output}} = \left.\begin{array}{l} (1-!_{"})\ (x_{\#})^{\times}(1-!_{,})\ (!_{.})\ \cdots\ (1-!_{0})\\ +\ (!_{"})\ (!_{\#})\ (!_{,})\ (1-!_{.})\ \cdots\ (!_{0})\\ +\ (!_{"})\ (1-!_{\#})(1-!_{,})\ (!_{.})\ \cdots\ (!_{0})\\ \quad\vdots\\ +\ (!_{"})\ (!_{\#})\ (1-!_{,})\ (!_{.})\ \cdots\ (!_{0})\end{array}\right\}$$

Corresponds to the TRUE rows in the truth table of the Boolean function

$EQ_{\text{ROBP}} \in$ BPP

Algorithm for $EQ_{\text{ROBP}} =$ "On input $\langle B_1, B_2\rangle$ [on variables $x_1, \dots, x_m$]
1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \dots, r_m$ where each $r_i \in \mathbb{F}_q$.
3. Evaluate B_1 and B_2 on r by using arithmetization.
4. If B_1 and B_2 agree on r then *accept*.
   If they disagree then *reject*."

Claim: (1) $B_1 \equiv B_2 \rightarrow \Pr[\, p_1(r) = p_2(r)\,] = 1$
(2) $B_1 \not\equiv B_2 \rightarrow \Pr[\, p_1(r) = p_2(r)\,] \leq {}^1/_3$

Proof (1): If $B_1 \equiv B_2$ then they agree on all Boolean inputs.
Thus their functions have the same truth table.
Thus their associated polynomials p_1 and p_2 are identical.
Thus p_1 and p_2 always agree (even on non-Boolean inputs).

Proof (2): If $B_1 \not\equiv B_2$ then $p_1 \neq p_2$ so $p = p_1 - p_2 \neq 0$.
From Schwartz-Zippel, $\Pr[\, p_1(r) = p_2(r)\,] \leq {}^{dm}/_q \leq {}^m/_{3m} = {}^1/_3$.
(Note that $d = 1$.)

Check-in 24.2
If the BPs were not read-once, the polynomials might have exponents ≥ 1. Where would the proof fail?
(a) $B_1 \equiv B_2$ implies they agree on all Boolean inputs
(b) Agreeing on all Boolean inputs implies $p_1 = p_2$
(c) Having $p_1 = p_2$ implies p_1 and p_2 always agree

p_1 and p_2 each have the form:

$$\begin{aligned}
&(1-x_1)\,(x_2)\,(1-x_3)\,(x_4)\cdots(1-x_m)\\
+\;&(x_1)\,(x_2)\,(x_3)\,(1-x_4)\cdots(x_m)\\
+\;&(x_1)\,(1-x_2)(1-x_3)\,(x_4)\cdots(x_m)\\
&\vdots\\
+\;&(x_1)\,(x_2)\,(1-x_3)\,(x_4)\cdots(x_m)
\end{aligned}$$

Check-in 24.2

$EQ_{\text{ROBP}} \in$ BPP

Algorithm for $EQ_{\text{ROBP}} =$ "On input $\langle B_1, B_2 \rangle$ [on variables $x_1, \dots, x_m$]

1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \dots, r_m$ where each $r_i \in \mathbb{F}_q$.
3. Evaluate B_1 and B_2 on r by using arithmetization.
4. If B_1 and B_2 agree on r then *accept*.
   If they disagree then *reject*."

Claim: (1) $B_1 \equiv B_2 \rightarrow \Pr[\, p_1(r) = p_2(r)\,] = 1$

(2) $B_1 \not\equiv B_2 \rightarrow \Pr[\, p_1(r) = p_2(r)\,] \leq {}^1\!/_3$

Proof (1): If $B_1 \equiv B_2$ then they agree on all Boolean inputs.
Thus their functions have the same truth table.
Thus their associated polynomials p_1 and p_2 are identical.
Thus p_1 and p_2 always agree (even on non-Boolean inputs).

Proof (2): If $B_1 \not\equiv B_2$ then $p_1 \neq p_2$ so $p = p_1 - p_2 \neq 0$.
From Schwartz-Zippel, $\Pr[\, p_1(r) = p_2(r)\,] \leq {}^{dm}\!/_q \leq {}^m\!/_{3m} = {}^1\!/_3$.
(Note that $d = 1$.)

Check-in 24.3

If p_1 and p_2 were exponentially large expressions, would that be a problem for the time complexity?

(a) Yes, but luckily they are polynomial in size.

(b) No, because we can evaluate them without writing them down.

p_1 and p_2 each have the form:

$$\begin{array}{llllll} & (1-x_1) & (x_2) & (1-x_3) & (x_4) & \cdots\ (1-x_m) \\ + & (x_1) & (x_2) & (x_3) & (1-x_4) & \cdots\ (x_m) \\ + & (x_1) & (1-x_2) & (1-x_3) & (x_4) & \cdots\ (x_m) \\ & & \vdots & & & \\ + & (x_1) & (x_2) & (1-x_3) & (x_4) & \cdots\ (x_m) \end{array}$$

Check-in 24.3

Quick review of today

1. Simulated Read-once Branching Programs by polynomials
2. Gave probabilistic polynomial equality testing method
3. Showed $!"_{\mathrm{ROBP}} \in \mathrm{BPP}$

MIT OpenCourseWare
https://ocw.mit.edu

18.404J / 18.4041J / 6.840J Theory of Computation
Fall 2020

For information about citing these materials or our Terms of Use, visit: https://ocw.mit.edu/terms.