18.404/6.840 Lecture 23

Last time:

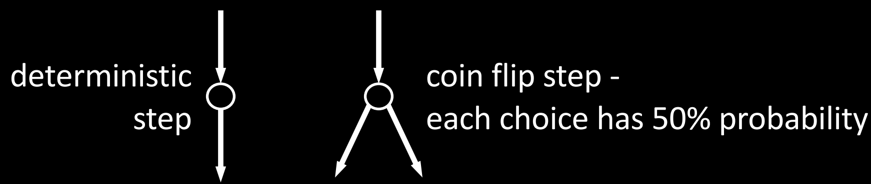
- $EQ_{\mathrm{REX}\uparrow}$ is EXPSPACE-complete
- Thus $EQ_{\mathrm{REX}\uparrow} \notin$ PSPACE
- Oracles and P versus NP

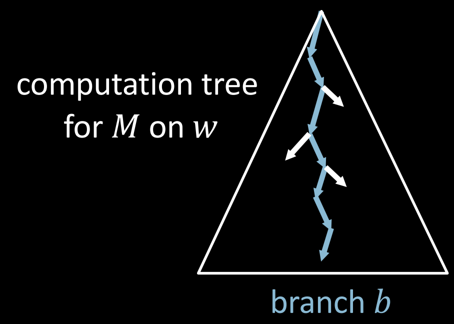
Today: (Sipser §10.2)

- Probabilistic computation
- The class BPP
- Branching programs

Probabilistic TMs

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





Pr[branch b] $= 2^{-k}$ where b has k coin flips

$$\Pr[\ M \text{ accepts } w\ ] = \sum_{b \text{ accepts}} \Pr[\text{ branch } b\ ]$$

Pr[M rejects w] $= 1 -$ Pr[M accepts w]

Defn: For $\epsilon \geq 0$ say PTM M decides language A with error probability ϵ if for every w, Pr[M gives the wrong answer about $w \in A$] $\leq \epsilon$

i.e., $w \in A \rightarrow$ Pr[M rejects w] $\leq \epsilon$

$w \notin A \rightarrow$ Pr[M accepts w] $\leq \epsilon$.

The Class BPP

Defn: BPP $= \{A \mid$ some poly-time PTM decides A with error $\epsilon = {}^1/_3\}$

Amplification lemma: If M_1 is a poly-time PTM with error $\epsilon_1 < {}^1/_2$ then, for any $0 < \epsilon_2 < {}^1/_2$, there is an equivalent poly-time PTM M_2 with error ϵ_2. Can strengthen to make $\epsilon_2 < 2^{-\mathrm{poly}(n)}$.

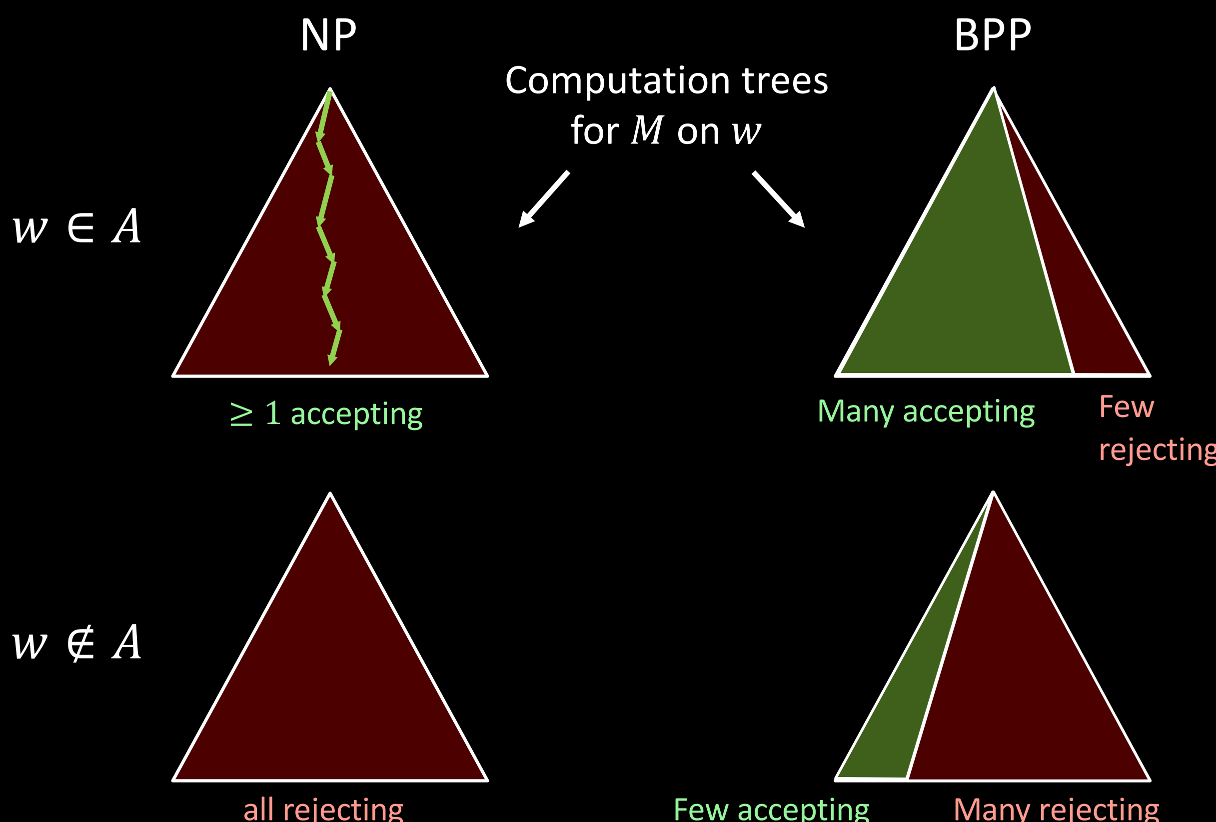
Proof idea: $M_2 =$ "On input w

1. Run M_1 on w for k times and output the majority response."

Details: Calculation to obtain k and the improved error probability.

Significance: Can make the error probability so small it is negligible.

NP and BPP



Check-in 23.1

Which of these are known to be true? Check all that apply.

(a) BPP is closed under union.

(b) BPP is closed under complement.

(c) $P \subseteq BPP$

(d) $BPP \subseteq PSPACE$

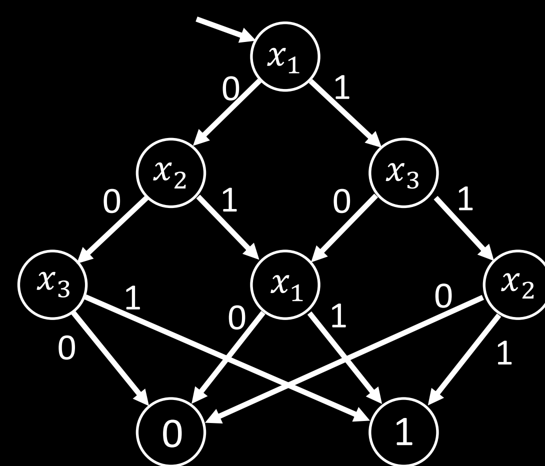
Check-in 23.1

Example: 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*.

BP B with query nodes $x_1, \dots, x_m$ describes a Boolean function $f\colon \{0,1\}^m \rightarrow \{0,1\}$:
Follow the path designated by the query nodes' outgoing edges
from the start note until reach an output node.



Example: For $x_1 = 1,\ x_2 = 0,\ x_3 = 1$

BPs are *equivalent* if they describe the same Boolean function.
Defn: $EQ_{\mathrm{BP}} = \{\langle B_1, B_2\rangle \mid B_1$ and B_2 are equivalent BPs (written $B_1 \equiv B_2$) $\}$

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

$EQ_{\mathrm{BP}} \in$ BPP ?
Instead, consider a restricted problem.

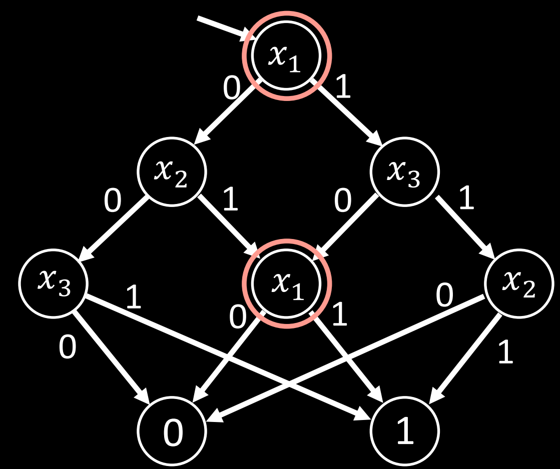
Read-once Branching Programs

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

> **Check-in 23.2**
> Assuming (as we will show) that $EQ_{\text{ROBP}} \in$ BPP, can we use that to show $EQ_{\text{BP}} \in$ BPP by converting branching programs to read-once branching programs?
>
> (a) Yes, there is no need to re-read inputs.
>
> (b) No, we cannot do that conversion in general.
>
> (c) No, the conversion is possible but not in polynomial-time.



Not read-once

Check-in 23.2

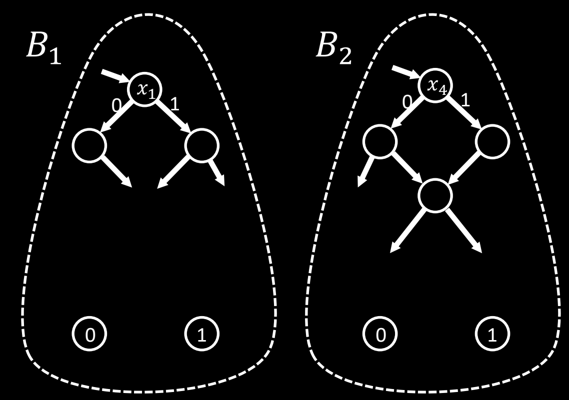
$EQ_{\text{ROBP}} \in \text{BPP}$

Theorem: $EQ_{\text{ROBP}} \in$ BPP
Proof attempt: Let M = "On input $\langle B_1, B_2\rangle$
1. Pick k random input assignments and evaluate B_1 and B_2 on each one.
2. If B_1 and B_2 ever disagree on those assignments then *reject*.
   If they always agree on those assignments then *accept*."

What k to chose?
If $B_1 \equiv B_2$ then they always agree so Pr[M accepts $\langle B_1, B_2\rangle$] = 1
If $B_1 \not\equiv B_2$ then want Pr[M accepts $\langle B_1, B_2\rangle$] $\leq 1/3$
so want Pr[M rejects $\langle B_1, B_2\rangle$] $\geq 2/3$.

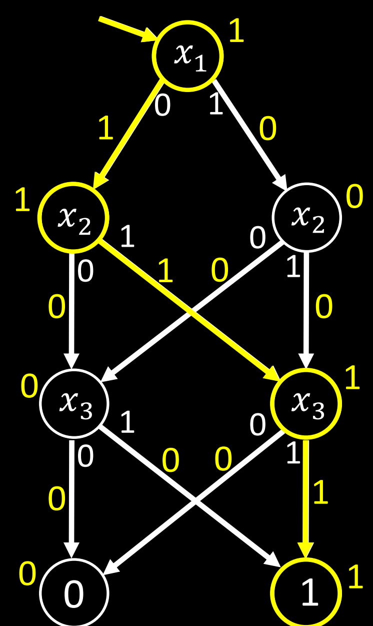
But B_1 and B_2 may disagree rarely, say in 1 of the 2^m possible assignments.
That would require exponentially many samples to have a good chance of finding a disagreeing assignment and thus would require $k > (2/3)2^m$.
But then this algorithm would use exponential time.

Try a different idea: Run B_1 and B_2 on non-Boolean inputs.



Boolean Labeling

Alternative way to view BP computation



Show by example: Input is $x_1 = 0,\ x_2 = 1,\ x_3 = 1$

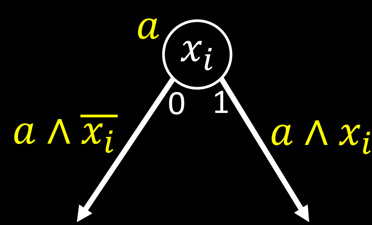
The BP follows its execution path.

Label all nodes and edges on the execution path with 1

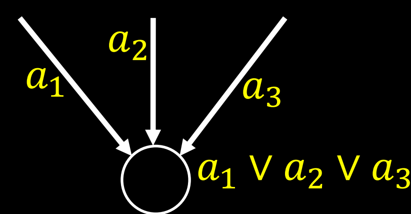
and off the execution path with 0.

Output the label of the output node 1.

Obtain the labeling inductively by using these rules:



Label edges from nodes

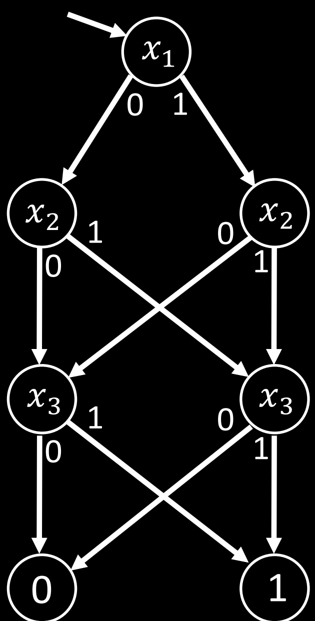


Label nodes from incoming edges

Arithmetization Method

Method: Simulate $\wedge$ and $\vee$ with $+$ and $\times$.

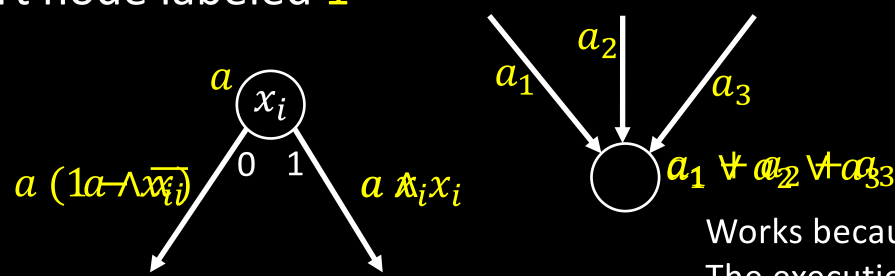
$$a \wedge b \;\rightarrow\; a \times b = ab$$
$$\bar{a} \;\rightarrow\; (1 - a)$$
$$a \vee b \;\rightarrow\; a + b - ab$$



Replace Boolean labeling with arithmetical labeling

Inductive rules:

Start node labeled 1

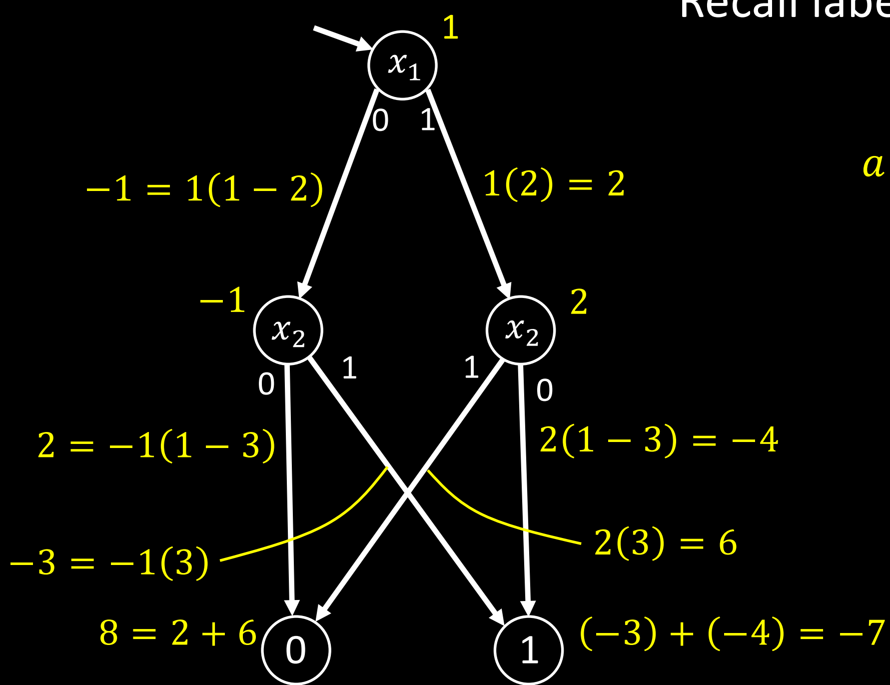


Works because the BP is acyclic.
The execution path can enter a node at most one time.

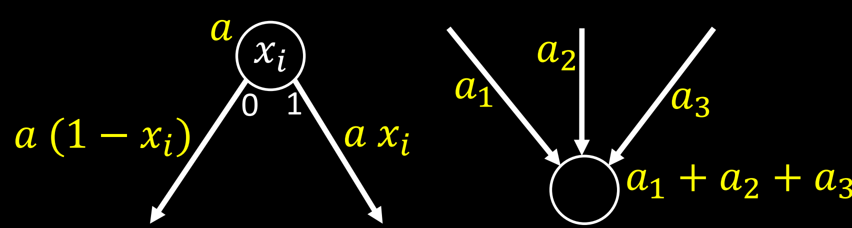
Non-Boolean Inputs

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

Example: $x_1 = 2,\ x_2 = 3$



Recall labeling rules:



Check-in 23.3

What is the output for this branching program using the arithmetized interpretation if $x_1 = 1,\ x_2 = y$?

(a) $(1 - y)$

(b) $(y + 1)$

(c) y

Check-in 23.3

Quick review of today

1. Defined probabilistic Turing machines
2. Defined the class BPP
3. Sketched the amplification lemma
4. Introduced branching programs and read-once branching programs
5. Started the proof that $EQ_{\text{ROBP}} \in$ BPP
6. Introduced the arithmetization method