بسم الله الرحمن الرحیم

نظریه علوم کامپیوتر

نظریه علوم کامپیوتر - بهار ۱۴۰۰-۱۴۰۱ - جلسه هفدهم: محاسبات تصادفی

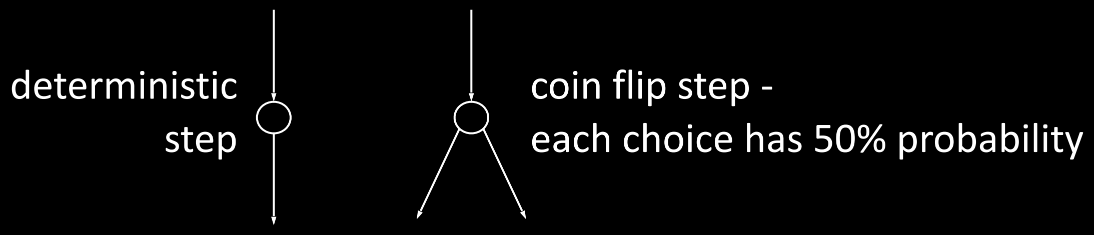
Theory of computation - 002 - S17 - BPP

Probabilistic TMs

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

Probabilistic TMs

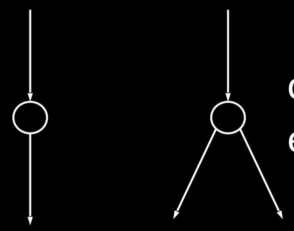
Defn: A probabilistic Turing machine (PTM) is a variant of a NTM where each computation step has 1 or 2 possible choices.



Probabilistic TMs

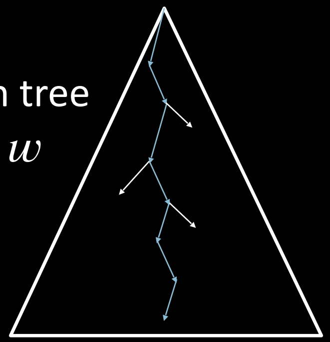
Defn: A <u>probabilistic Turing machine</u> (PTM) is a variant of a NTM where each computation step has 1 or 2 possible choices.

deterministic step



coin flip step - each choice has 50% probability

computation tree for M on w

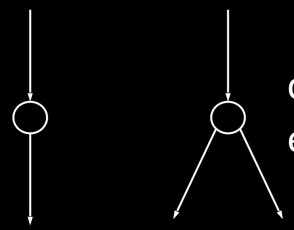


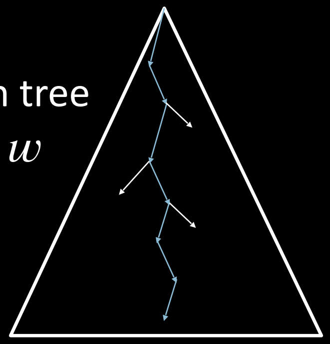
branch b

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

Probabilistic TMs

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



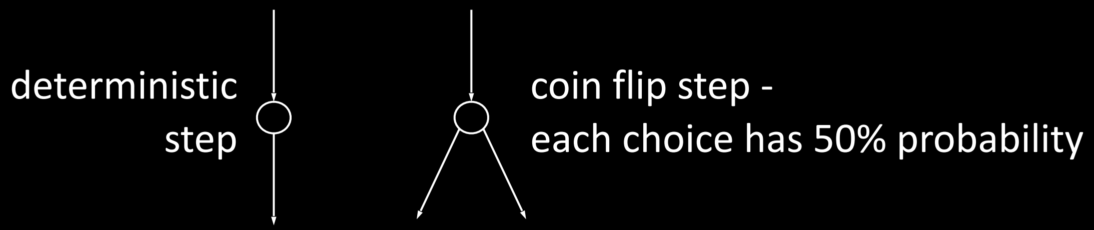


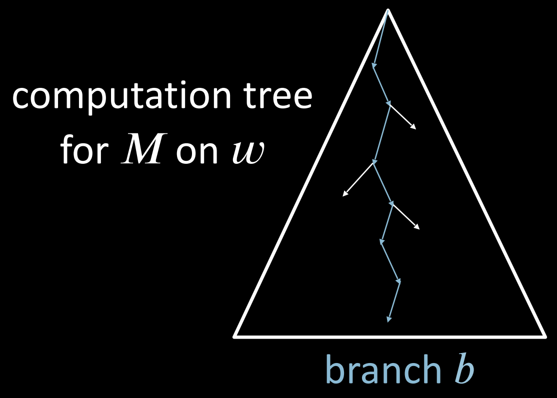
Pr[branch b] $= 2^{-k}$ where b has k coin flips

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

Probabilistic TMs

Defn: A <u>probabilistic Turing machine</u> (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

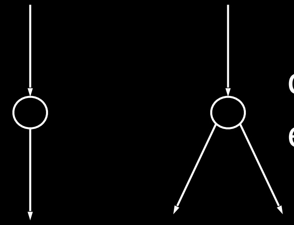
Pr[M accepts w] $= \sum_{b \text{ accepts}}$ Pr[branch b]

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

Probabilistic TMs

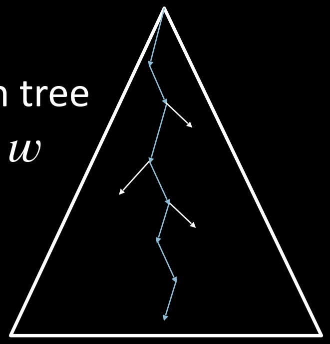
Defn: A <u>probabilistic Turing machine</u> (PTM) is a variant of a NTM where each computation step has 1 or 2 possible choices.

deterministic step



coin flip step - each choice has 50% probability

computation tree for M on w



branch b

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

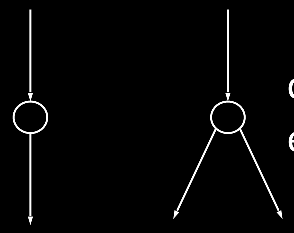
Pr[M accepts w] $= \sum_{b \text{ accepts}}$ Pr[branch b]

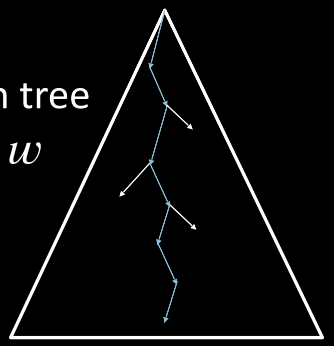
Pr[M rejects w] $= 1 -$ Pr[M accepts w]

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

Probabilistic TMs

Defn: A <u>probabilistic Turing machine</u> (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 accepts w] $= \sum_{b \text{ accepts}}$ Pr[branch b]

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

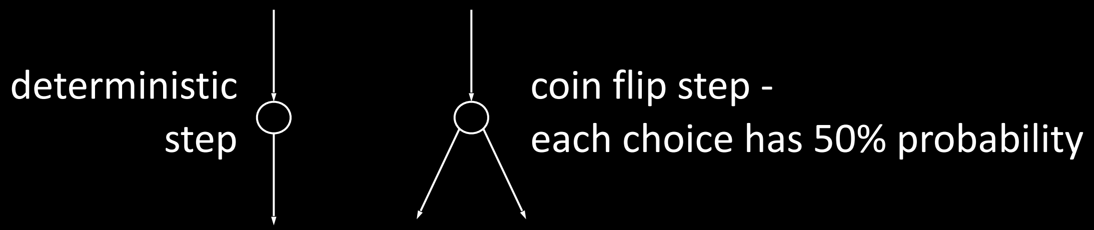
Defn: For $\epsilon \geq 0$ say PTM M <u>decides language A with error probability ϵ</u> if for every w, Pr[M gives the wrong answer about $w \in A$] $\leq \epsilon$

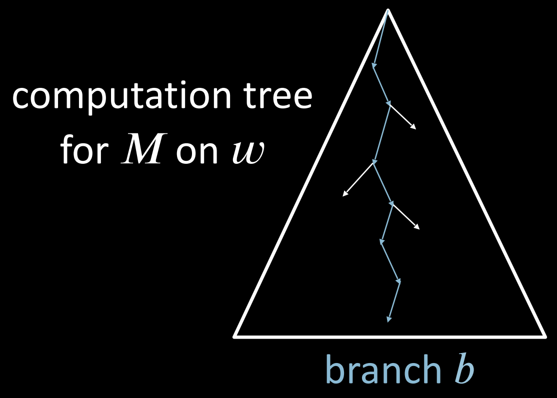
i.e., $w \in A \rightarrow$ Pr[M rejects w] $\leq \epsilon$

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

Probabilistic TMs

Defn: A <u>probabilistic Turing machine</u> (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 accepts w] $= \sum_{b \text{ accepts}}$ Pr[branch b]

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

Defn: For $\epsilon \geq 0$ say PTM M <u>decides language A with error probability ϵ</u> 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 $= \{A \,|\,$ some poly-time PTM decides A with error $\epsilon = \frac{1}{3}\}$

Amplification lemma: If M_1 is a poly-time PTM with error $\epsilon_1 < \frac{1}{2}$ then,

for any $0 < \epsilon_2 < \frac{1}{2}$, there is an equivalent poly-time PTM M_2 with error ϵ_2.

Can strengthen to make $\epsilon_2 < 2^{-\text{poly}(n)}$.

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.

The Class BPP

Defn: BPP $= \{A \,\big|$ some poly-time PTM decides A with error $\epsilon = \dfrac{1}{3}\}$

Amplification lemma: If M_1 is a poly-time PTM with error $\epsilon_1 < \dfrac{1}{2}$ then,

for any $0 < \epsilon_2 < \dfrac{1}{2}$, 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."

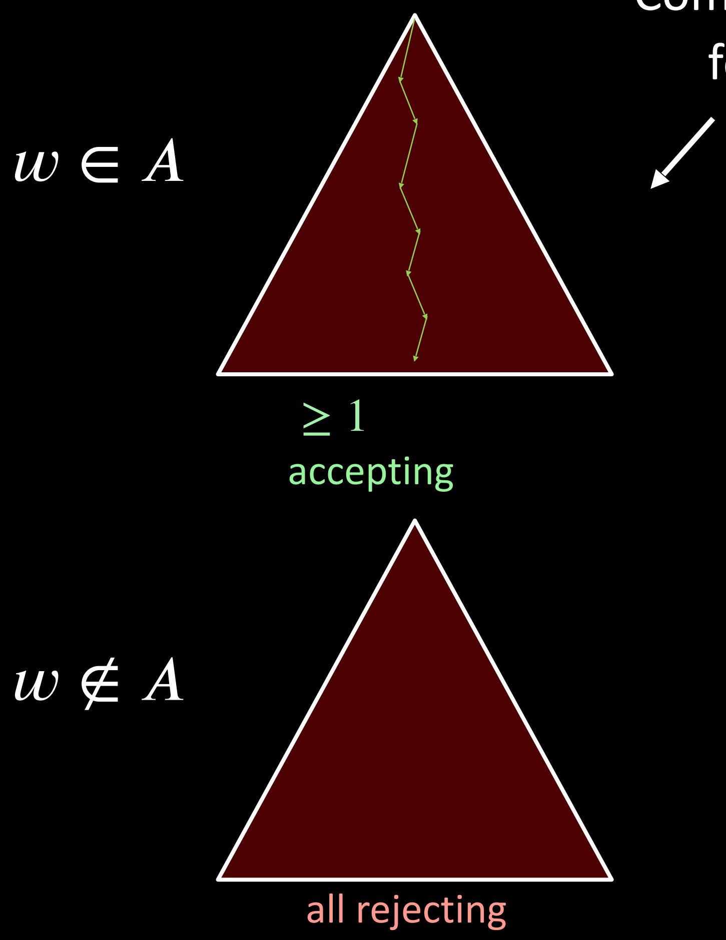
Details: Calculation to obtain k and the improved error probability.

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

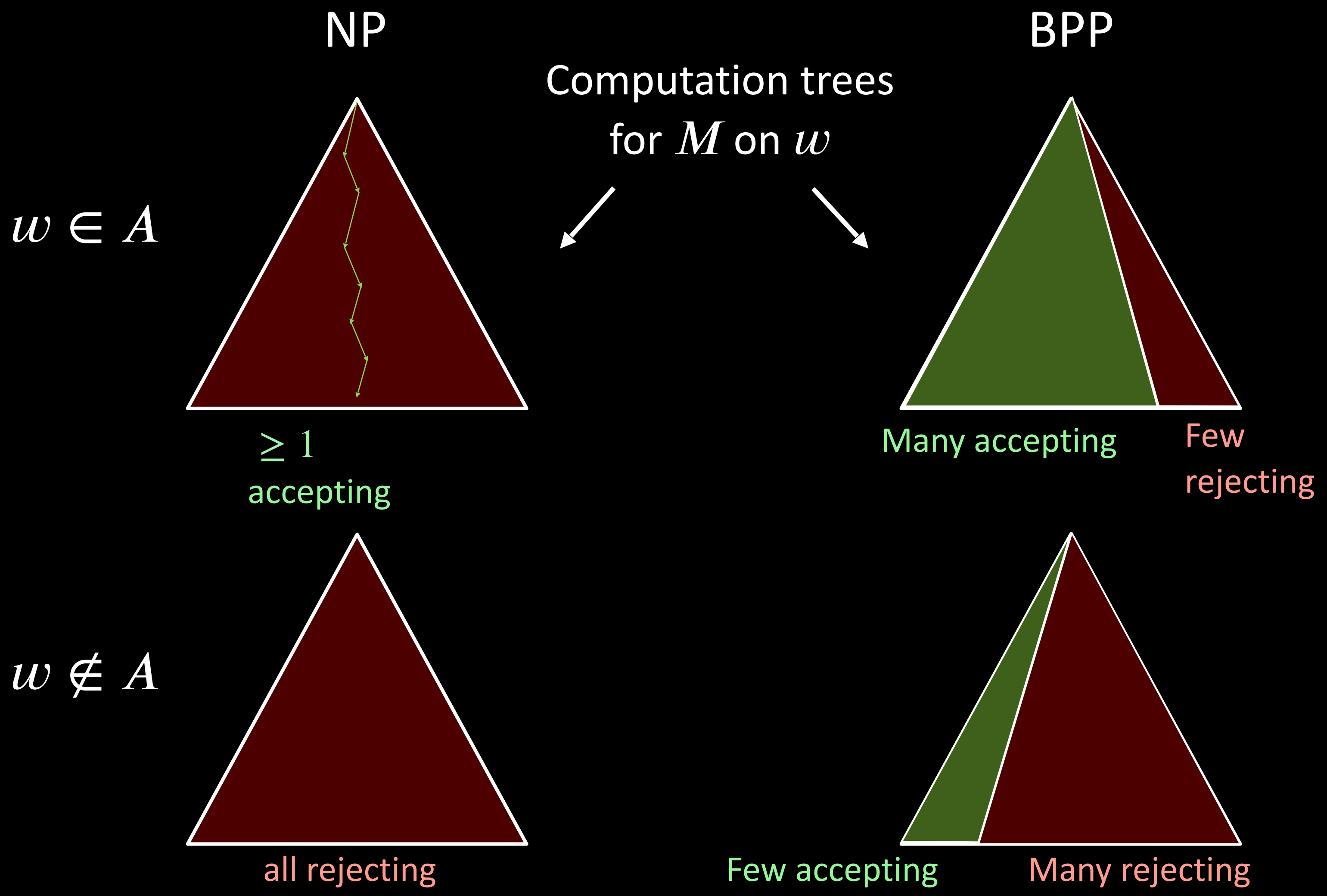
NP and BPP

NP

Computation trees for M on w

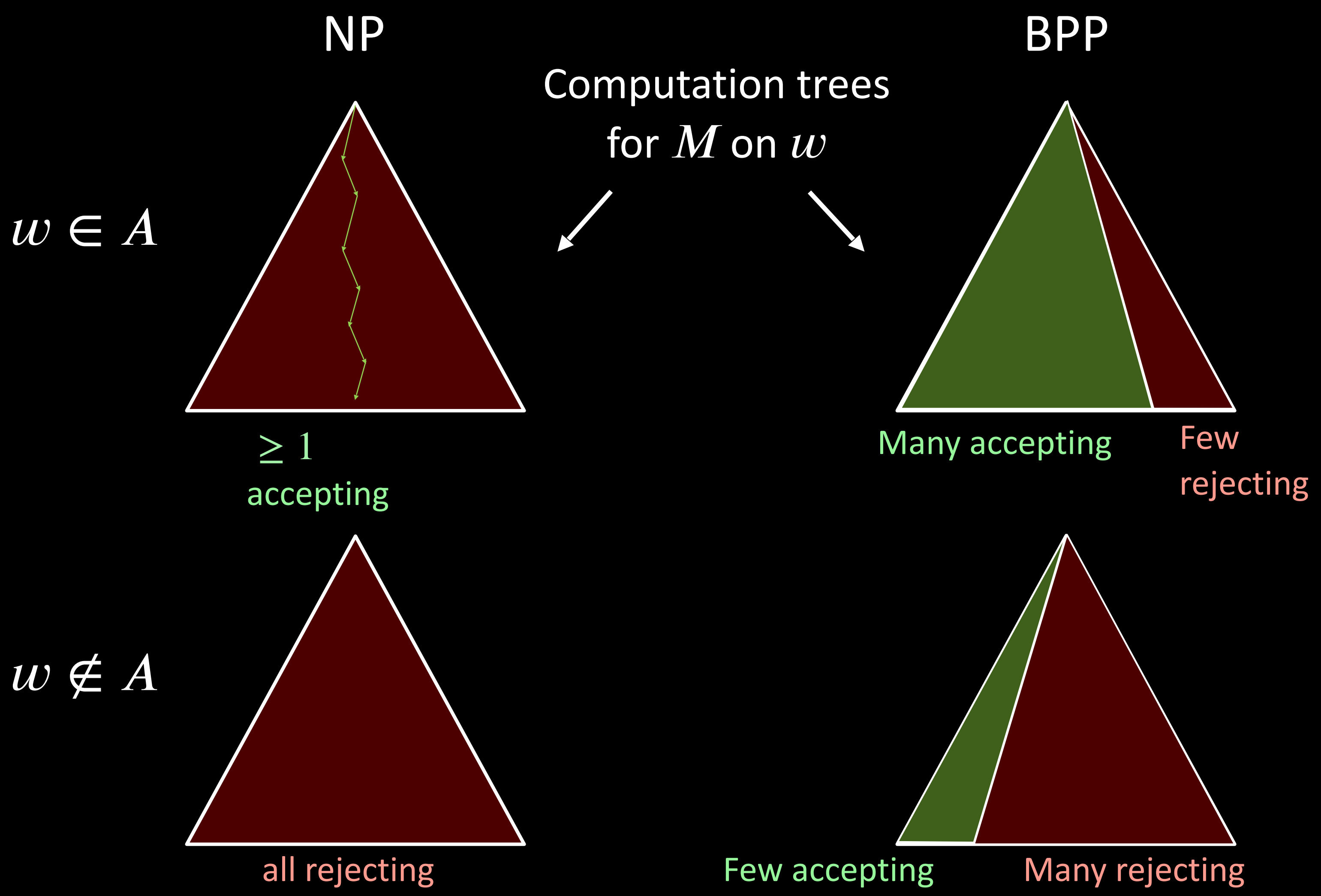


NP and BPP



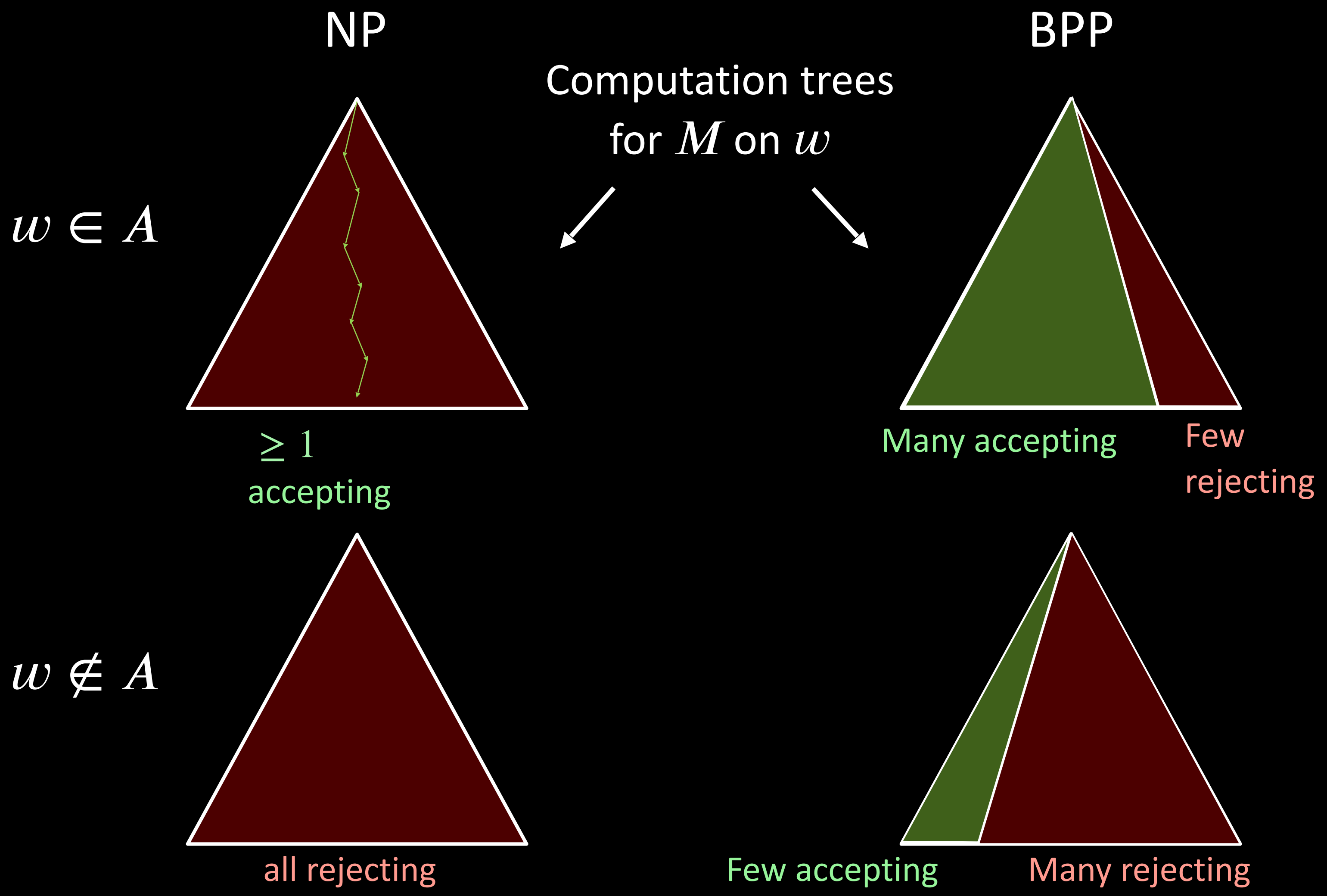
NP
BPP
Computation trees
for M on w
$w \in A$
$w \notin A$
≥ 1
accepting
Many accepting
Few
rejecting
all rejecting
Few accepting
Many rejecting

NP and BPP



Check-in 23.1

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

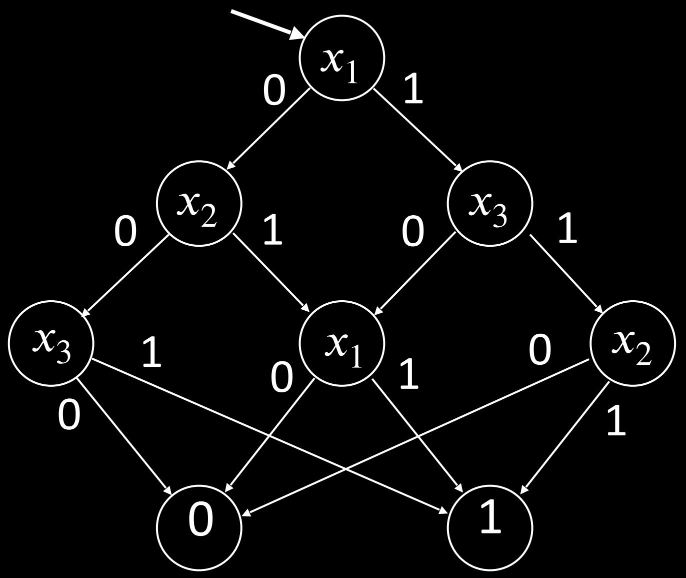
(d) BPP $\subseteq$ PSPACE

Check-in 23.1

Example: Branching Programs

Defn: A branching program (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*.

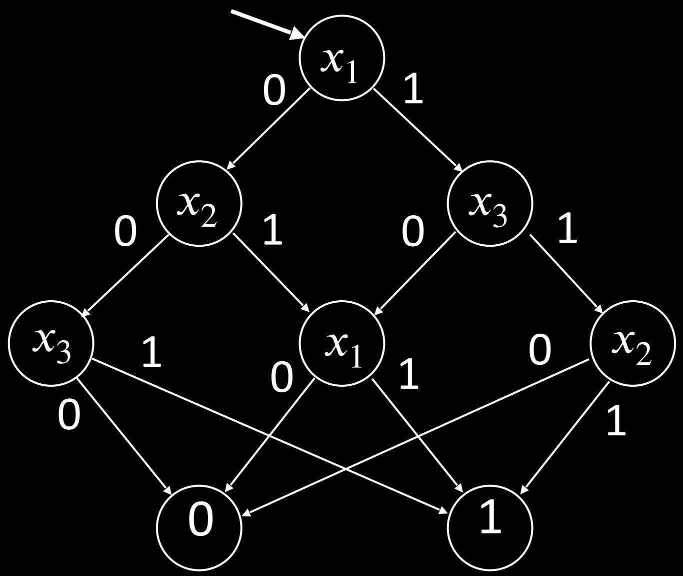


Example: Branching Programs

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



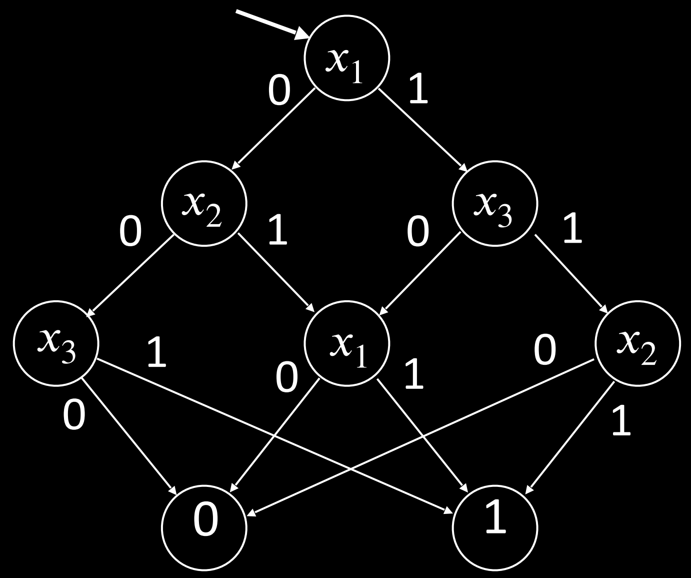
Example: Branching Programs

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

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



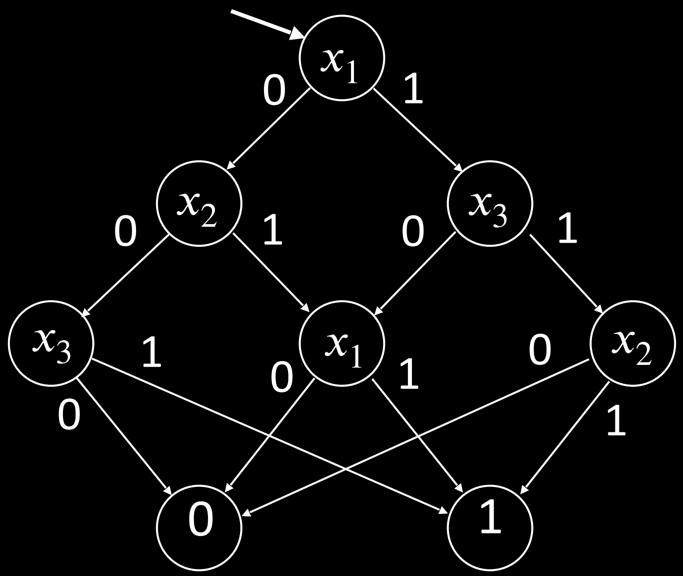
Example: Branching Programs

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

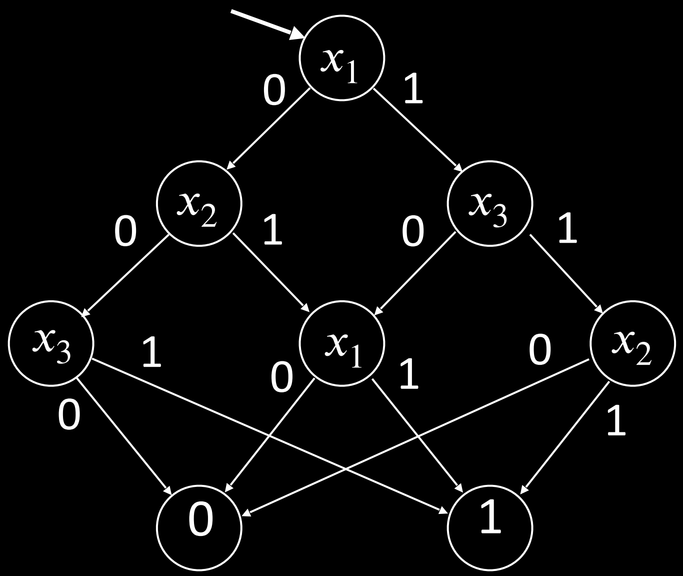
Example: For $x_1 = 1,\ \ x_2 = 0,\ \ x_3 = 1$ we have $f(101) = 0 =$ output.



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

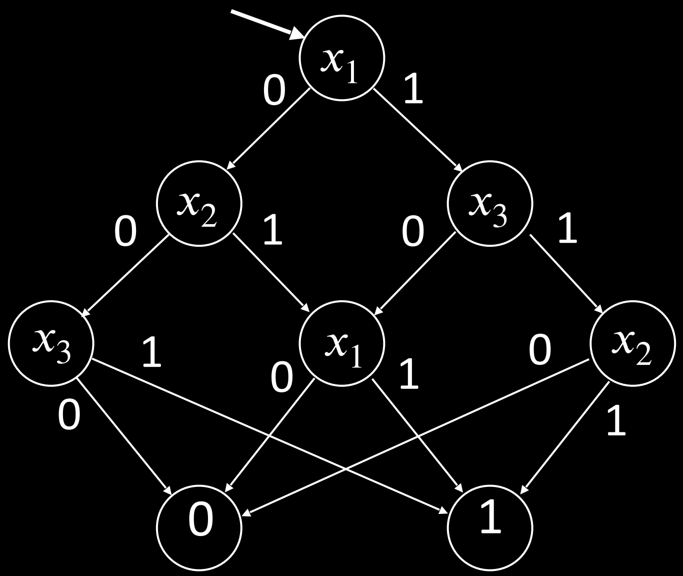
Example: For $x_1 = 1$, $x_2 = 0$, $x_3 = 1$ we have $f(101) = 0 =$ output.

BPs are *equivalent* if they describe the same Boolean function.

Example: Branching Programs

Defn: A branching program (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 $x_1, \ldots, x_m$ describes a Boolean function $f\colon\{0,1\}^m \to \{0,1\}$:
Follow the path designated by the query nodes' outgoing edges from the start note until reach an output node.

Example: For $x_1 = 1$, $x_2 = 0$, $x_3 = 1$ we have $f(101) = 0 =$ output.

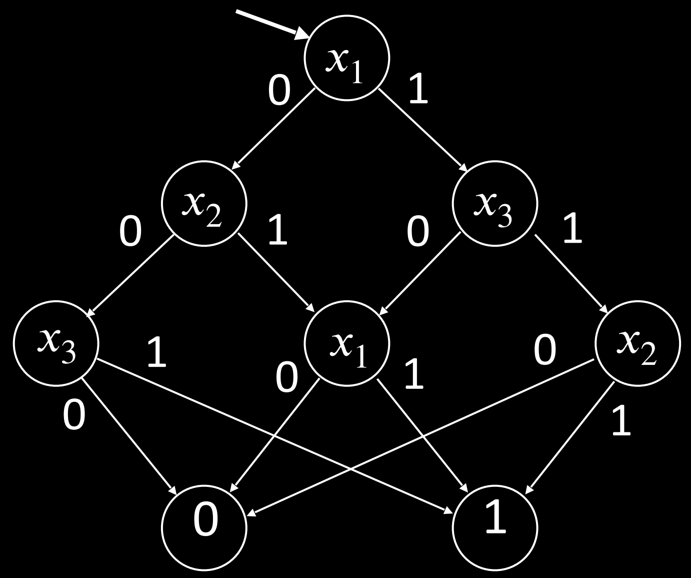
BPs are *equivalent* if they describe the same Boolean function.

Defn: EQ**BP** $= \{\langle B_1, B_2\rangle \mid B_1$ and B_2 are equivalent BPs (written $B_1 \equiv B_2$) $\}$

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

Example: For $x_1 = 1$, $x_2 = 0$, $x_3 = 1$ we have $f(101) = 0 =$ output.

BPs are *equivalent* if they describe the same Boolean function.

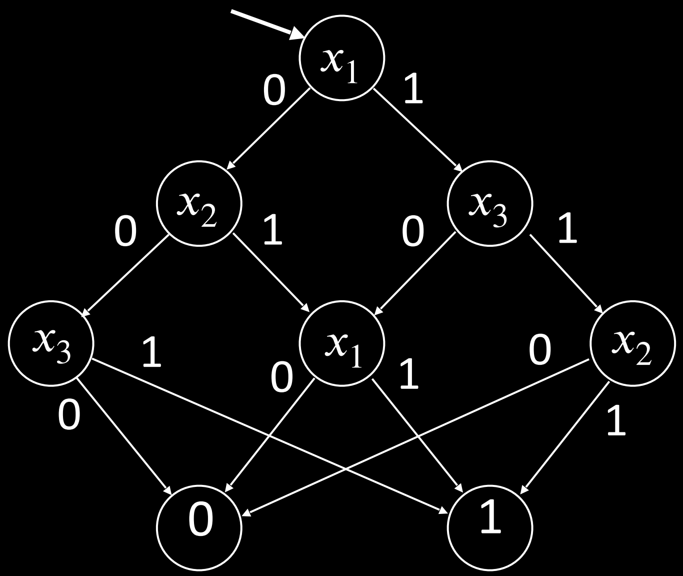
Defn: EQ**BP** $= \{\langle B_1, B_2\rangle \mid B_1$ and B_2 are equivalent BPs (written $B_1 \equiv B_2$) $\}$

Theorem: EQ**BP** is coNP-complete (on pset 6)

Example: Branching Programs

Defn: A branching program (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 $x_1, \ldots, x_m$ describes a Boolean function $f: \{0,1\}^m \rightarrow \{0,1\}$:
Follow the path designated by the query nodes' outgoing edges from the start note until reach an output node.

Example: For $x_1 = 1, \;\; x_2 = 0, \;\; x_3 = 1$ we have $f(101) = 0 =$ output.

BPs are *equivalent* if they describe the same Boolean function.

Defn: EQ**BP** $= \{\langle B_1, B_2\rangle \mid B_1$ and B_2 are equivalent BPs (written $B_1 \equiv B_2$) $\}$

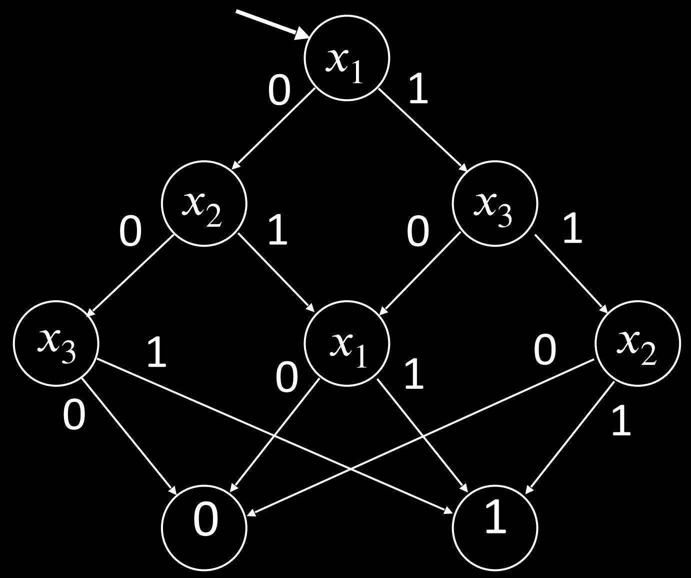
Theorem: EQ**BP** is coNP-complete (on pset 6)

$EQ\text{BP} \in$ BPP ?

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

Example: For $x_1 = 1$, $x_2 = 0$, $x_3 = 1$ we have $f(101) = 0 =$ output.

BPs are *equivalent* if they describe the same Boolean function.

Defn: EQ**BP** $= \{\langle B_1, B_2\rangle \mid B_1$ and B_2 are equivalent BPs (written $B_1 \equiv B_2$) $\}$

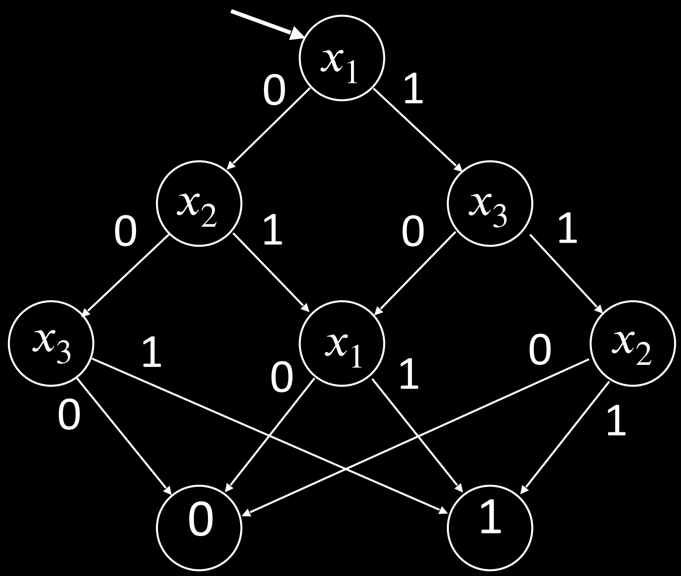
Theorem: EQ**BP** is coNP-complete (on pset 6)

$EQ\mathrm{BP} \in$ BPP ? Unknown. That would imply NP $\subseteq$ BPP and would be surprising!

Example: Branching Programs

Defn: A branching program (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 $x_1, \ldots, x_m$ describes a Boolean function $f: \{0,1\}^m \to \{0,1\}$:
Follow the path designated by the query nodes' outgoing edges from the start note until reach an output node.

Example: For $x_1 = 1$, $x_2 = 0$, $x_3 = 1$ we have $f(101) = 0 =$ output.

BPs are *equivalent* if they describe the same Boolean function.

Defn: EQ**BP** $= \{\langle B_1, B_2\rangle \mid B_1$ and B_2 are equivalent BPs (written $B_1 \equiv B_2$) $\}$

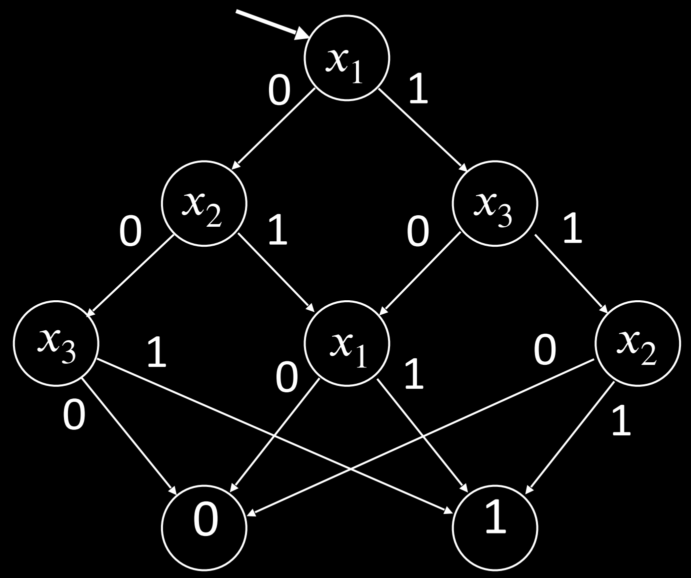
Theorem: EQ**BP** is coNP-complete (on pset 6)

$EQ\text{BP} \in$ BPP ? Unknown. That would imply NP $\subseteq$ BPP and would be surprising!
Instead, consider a restricted problem.

Example: Branching Programs

Defn: A branching program (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 $x_1, \ldots, x_m$ describes a Boolean function $f : \{0,1\}^m \to \{0,1\}$:
Follow the path designated by the query nodes' outgoing edges from the start note until reach an output node.

Example: For $x_1 = 1$, $x_2 = 0$, $x_3 = 1$ we have $f(101) = 0 =$ output.

BPs are *equivalent* if they describe the same Boolean function.

Defn: EQ**BP** $= \{\langle B_1, B_2 \rangle \mid B_1$ and B_2 are equivalent BPs (written $B_1 \equiv B_2$) $\}$

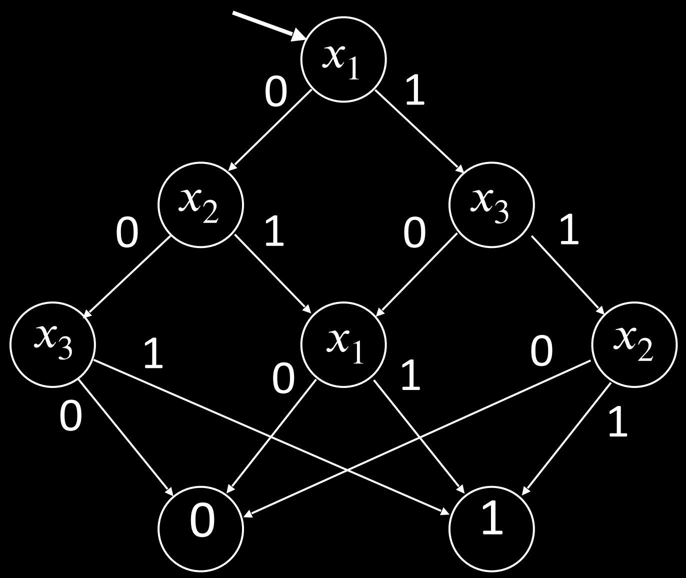
Theorem: EQ**BP** is coNP-complete (on pset 6)

$EQ\text{BP} \in$ BPP ? Unknown. That would imply NP $\subseteq$ BPP and would be surprising!
Instead, consider a restricted problem.

Read-once Branching Programs

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

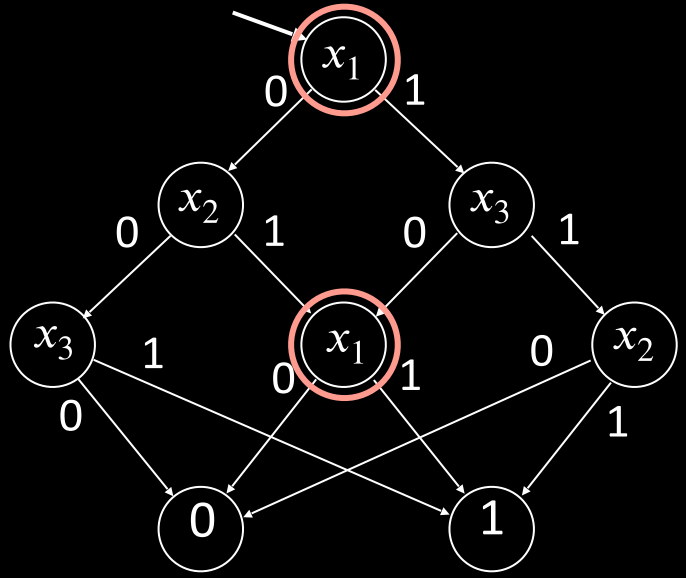
Defn: EQ**ROBP** $= \{\langle B_1, B_2 \rangle \mid B_1$ and B_2 are equivalent read-once BPs$\}$



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**ROBP** $= \{\langle B_1, B_2\rangle \mid B_1$ and B_2 are equivalent read-once BPs$\}$



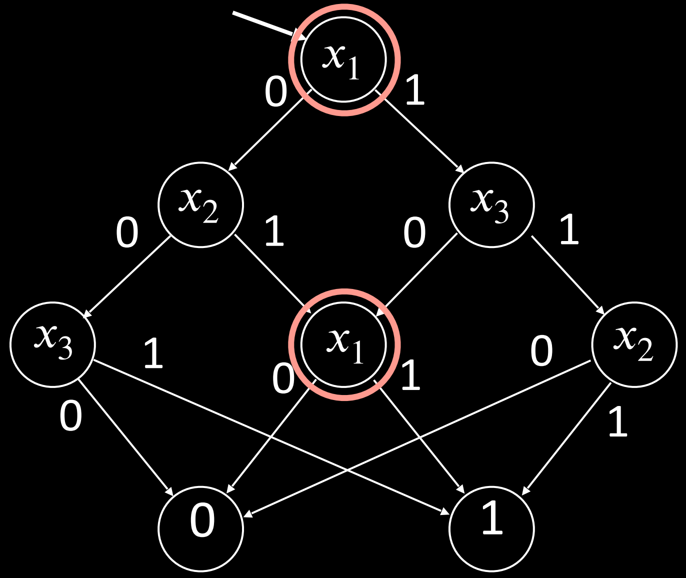
Not read-once

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**ROBP** $= \{\langle B_1, B_2\rangle \mid B_1$ and B_2 are equivalent read-once BPs$\}$

Theorem: EQ**ROBP** $\in$ BPP



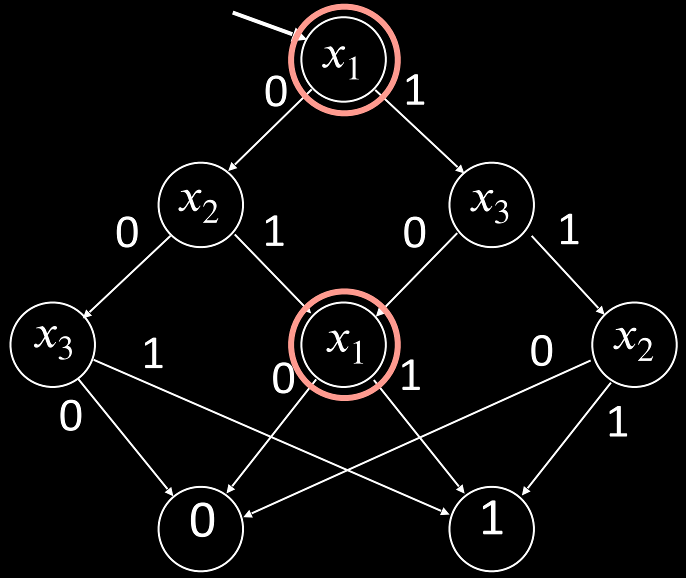
Not read-once

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**ROBP** $= \{\langle B_1, B_2\rangle \mid B_1$ and B_2 are equivalent read-once BPs$\}$

Theorem: EQ**ROBP** $\in$ BPP



Not read-once

Check-in 23.2

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**ROBP** $= \{\langle B_1, B_2\rangle \mid B_1$ and B_2 are equivalent read-once BPs$\}$

Theorem: EQ**ROBP** $\in$ BPP

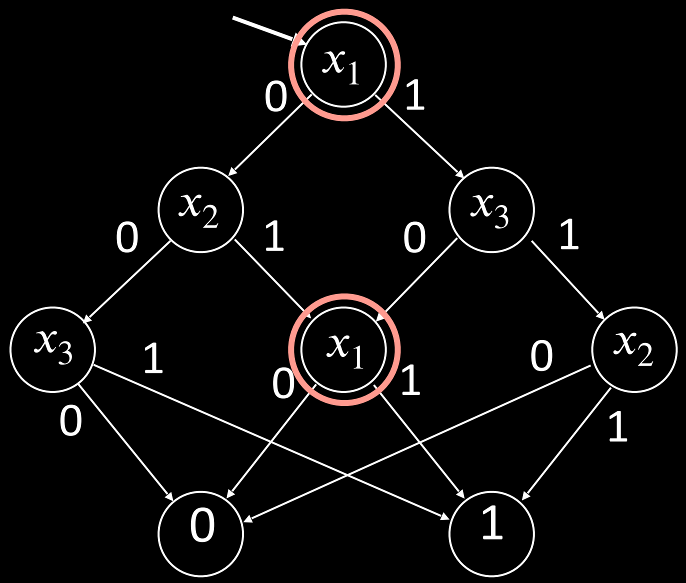
Check-in 23.2

Assuming (as we will show) that EQROBP $\in$ BPP, can we use that to show EQBP $\in$ 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

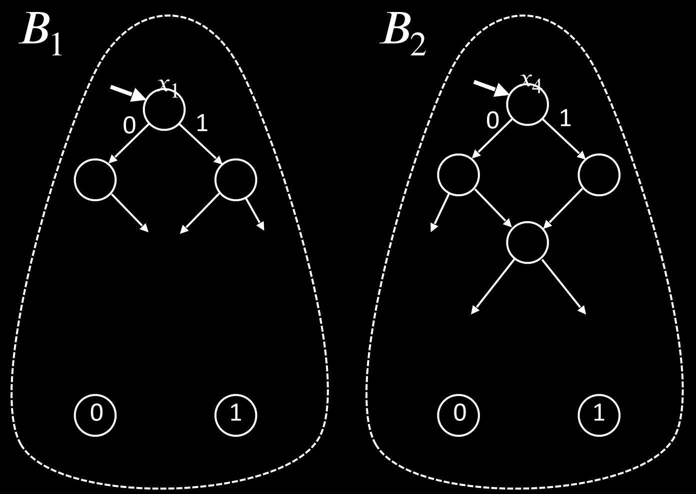
EQROBP $\in$ BPP

Theorem: EQ**ROBP** $\in$ BPP

EQROBP $\in$ BPP

Theorem: EQ**ROBP** $\in$ BPP

Proof attempt: Let M = "On input $\langle B_1, B_2 \rangle$

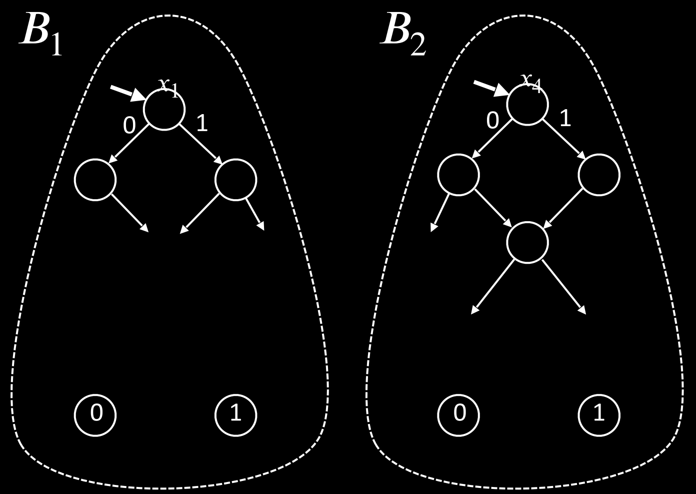


EQROBP $\in$ BPP

Theorem: EQ**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.



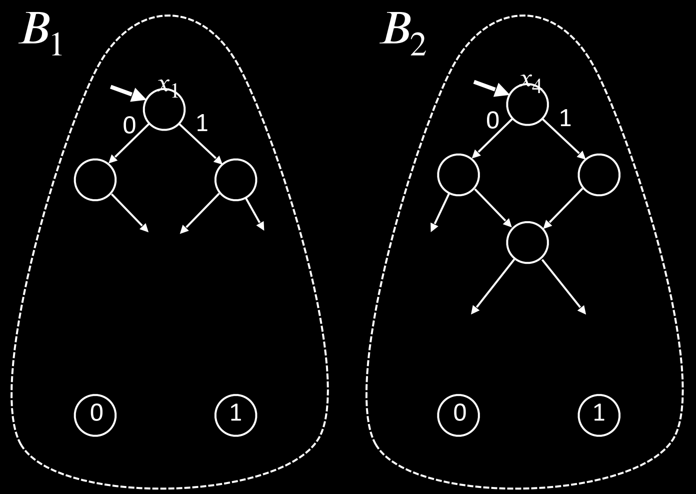
$EQ_{\text{ROBP}} \in$ BPP

Theorem: $EQ_{\textbf{ROBP}} \in$ 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*."



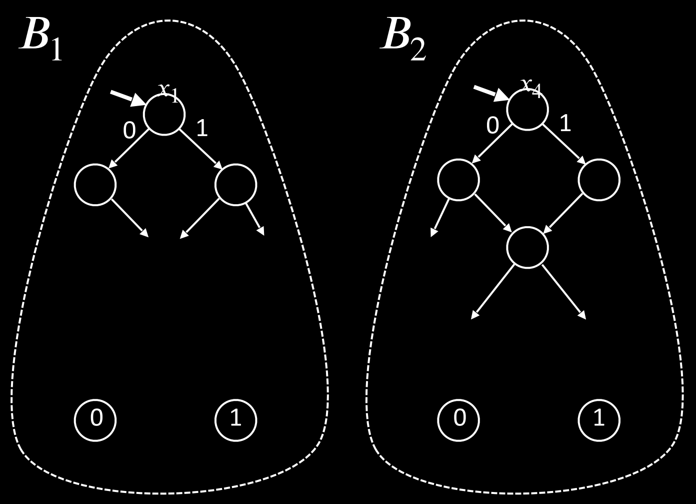
EQROBP $\in$ BPP

Theorem: EQ**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?

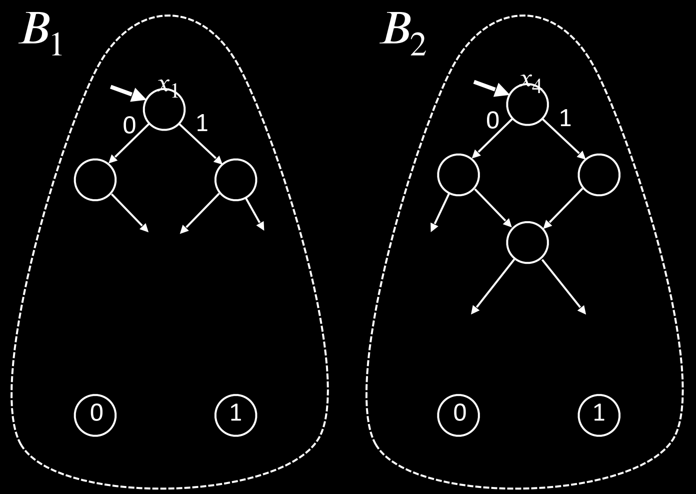


EQROBP $\in$ BPP

Theorem: EQ**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?

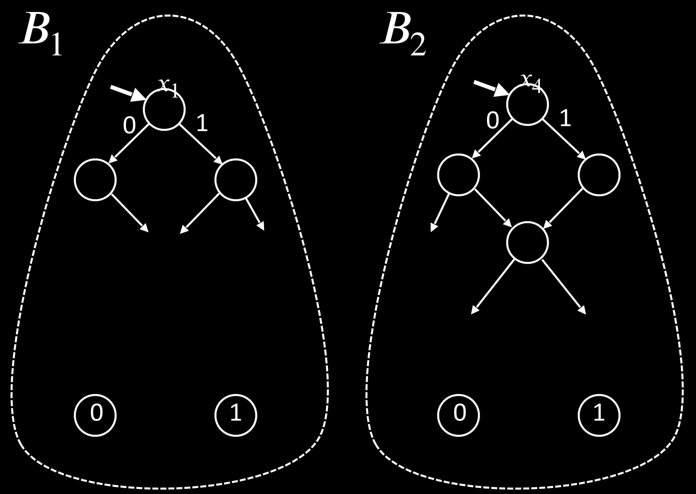
If $B_1 \equiv B_2$ then they always agree so Pr[M accepts $\langle B_1, B_2\rangle$] = 1

EQROBP $\in$ BPP

Theorem: EQ**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

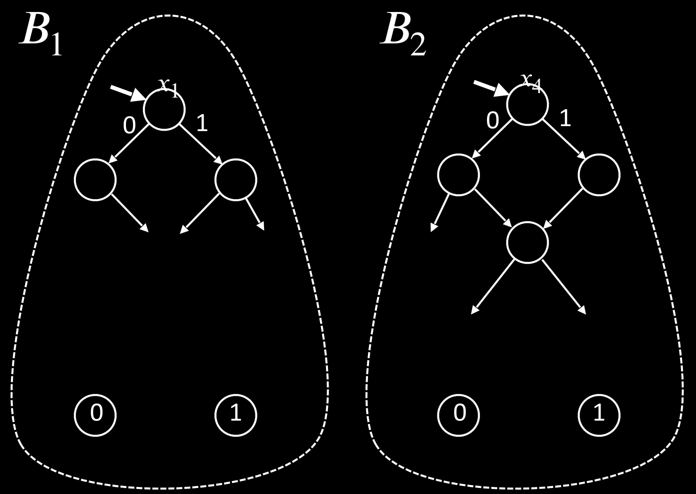
If $B_1 \not\equiv B_2$ then want Pr[M accepts $\langle B_1, B_2 \rangle$] $\leq \frac{1}{3}$

EQROBP $\in$ BPP

Theorem: EQ**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 $\langle B_1, B_2\rangle$] $\leq \frac{1}{3}$

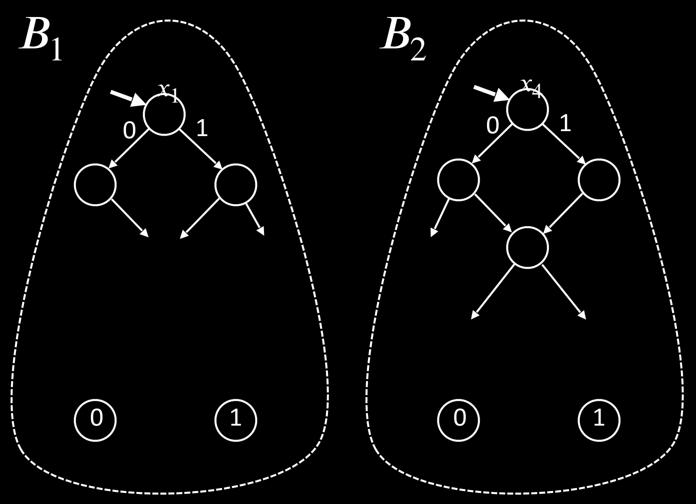
so want Pr[M rejects $\langle B_1, B_2\rangle$] $\geq \frac{2}{3}$.

EQROBP $\in$ BPP

Theorem: EQ**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 $\langle B_1, B_2 \rangle$] $\leq \frac{1}{3}$

so want Pr[M rejects $\langle B_1, B_2 \rangle$] $\geq \frac{2}{3}$.

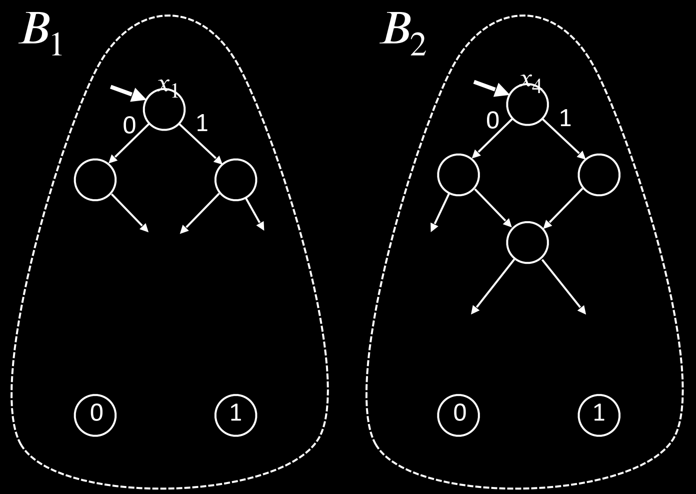
But B_1 and B_2 may disagree rarely, say in 1 of the 2^m possible assignments.

$EQ_{\text{ROBP}} \in$ 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 $\langle B_1, B_2 \rangle$] $\leq \frac{1}{3}$

so want Pr[M rejects $\langle B_1, B_2 \rangle$] $\geq \frac{2}{3}$.

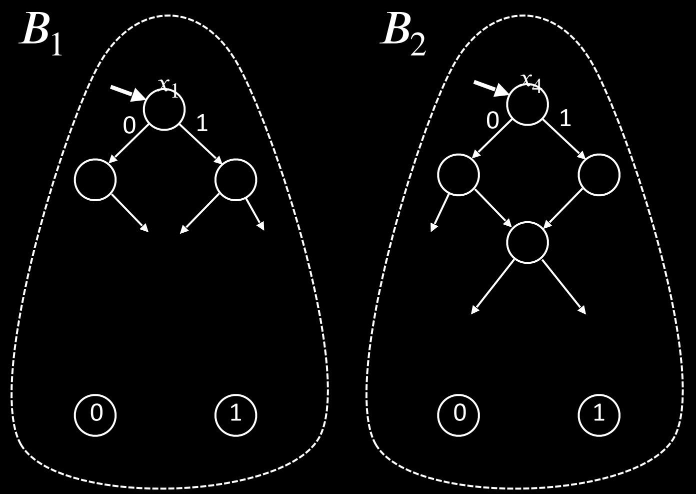
But B_1 and B_2 may disagree rarely, say in 1 of the 2^m possible assignments. That would require exponentially many samples to have a good chance of finding a disagreeing assignment and thus would require $k > \left(\frac{2}{3}\right)2^m$.

$EQ_{\text{ROBP}} \in$ 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 $\langle B_1, B_2 \rangle$] $\leq \frac{1}{3}$

so want Pr[M rejects $\langle B_1, B_2 \rangle$] $\geq \frac{2}{3}$.

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 $k > \left(\frac{2}{3}\right)2^m$.

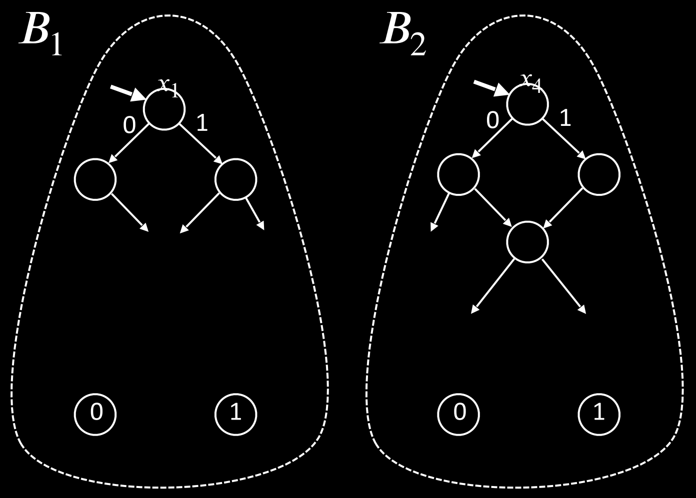
But then this algorithm would use exponential time.

EQROBP $\in$ BPP

Theorem: EQ**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 $\langle B_1, B_2\rangle$] $\leq \frac{1}{3}$

so want Pr[M rejects $\langle B_1, B_2\rangle$] $\geq \frac{2}{3}$.

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 $k > \left(\frac{2}{3}\right)2^m$.

But then this algorithm would use exponential time.

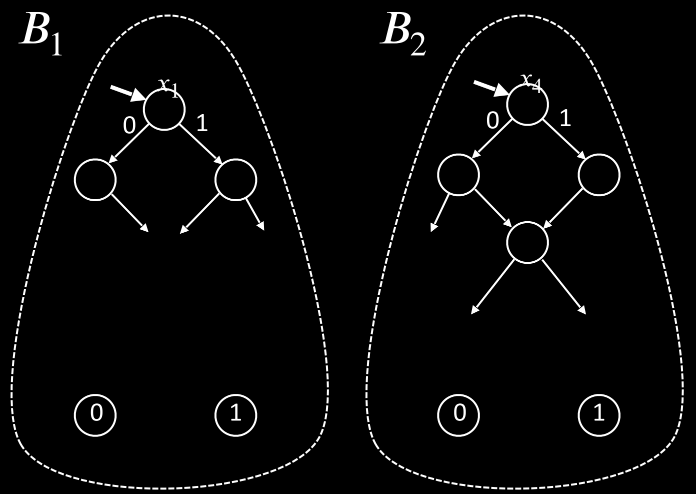
Try a different idea: Run B_1 and B_2 on <u>non-Boolean inputs</u>.

$EQ_{\text{ROBP}} \in$ 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 $\langle B_1, B_2\rangle$] $\leq \frac{1}{3}$

so want Pr[M rejects $\langle B_1, B_2\rangle$] $\geq \frac{2}{3}$.

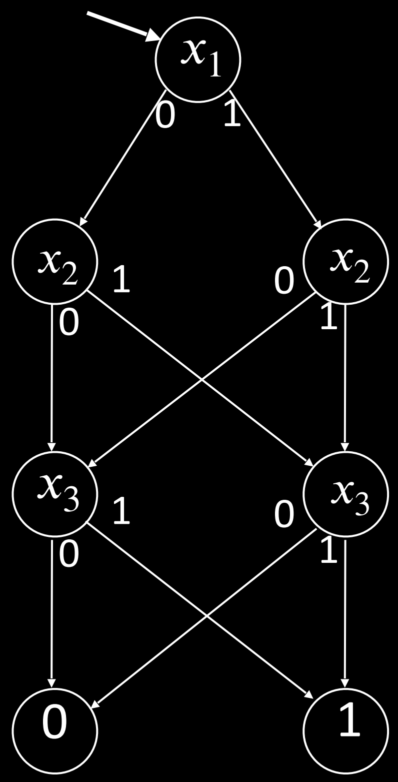
But B_1 and B_2 may disagree rarely, say in 1 of the 2^m possible assignments.
That would require exponentially many samples to have a good chance of finding a disagreeing assignment and thus would require $k > \left(\frac{2}{3}\right)2^m$.

But then this algorithm would use exponential time.

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

Boolean Labeling

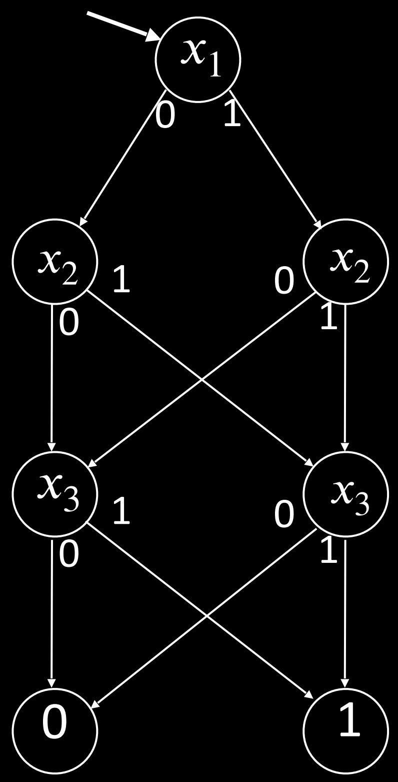
Alternative way to view BP computation



Boolean Labeling

Alternative way to view BP computation

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

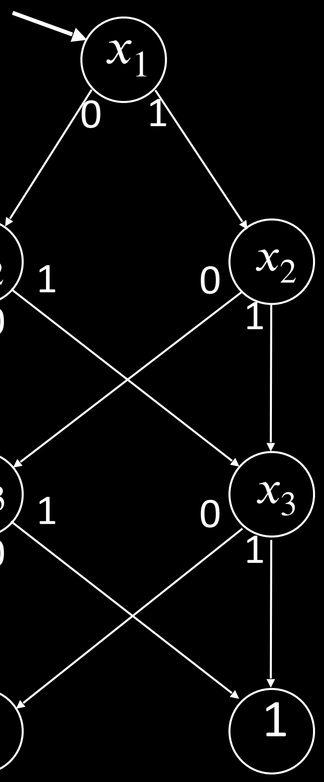


Boolean Labeling

Alternative way to view BP computation

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

The BP follows its execution path.

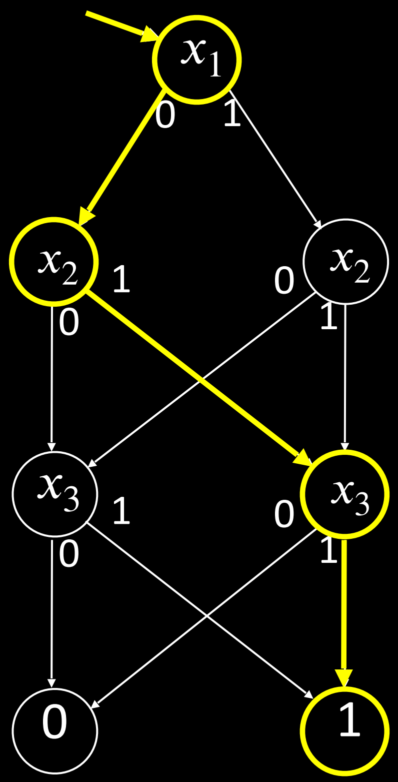


Boolean Labeling

Alternative way to view BP computation

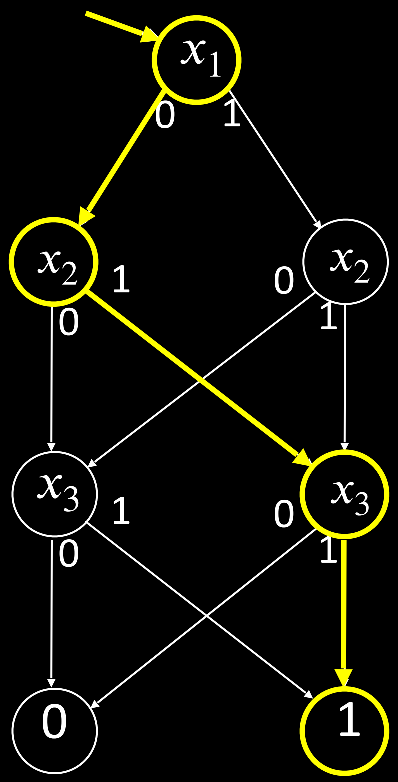
Show by example: Input is $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$

The BP follows its execution path.



Boolean Labeling

Alternative way to view BP computation



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

The BP follows its execution path.

Label all nodes and edges on the execution path with 1

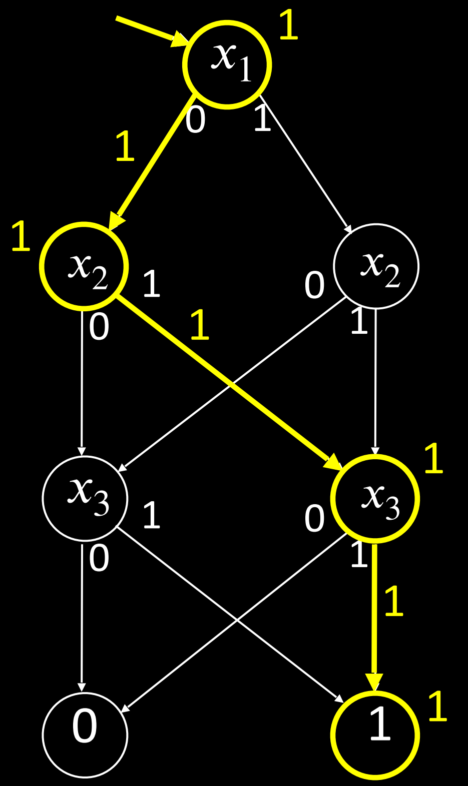
Boolean Labeling

Alternative way to view BP computation

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

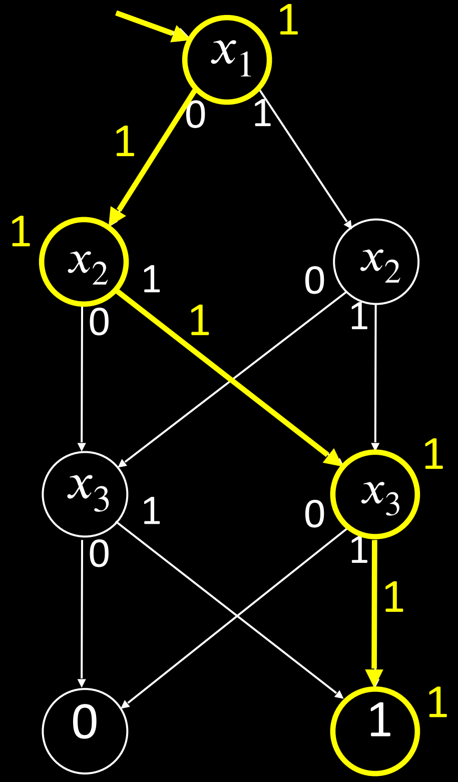
The BP follows its execution path.

Label all nodes and edges on the execution path with 1



Boolean Labeling

Alternative way to view BP computation



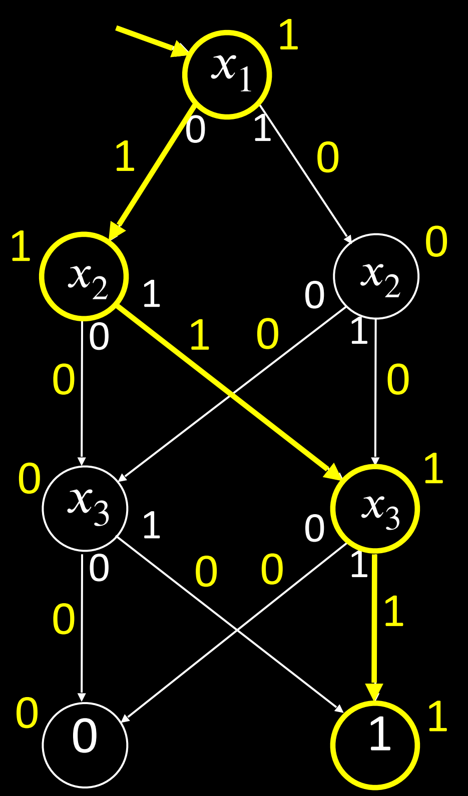
Show by example: Input is $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$

The BP follows its execution path.

Label all nodes and edges on the execution path with 1 and off the execution path with 0.

Boolean Labeling

Alternative way to view BP computation



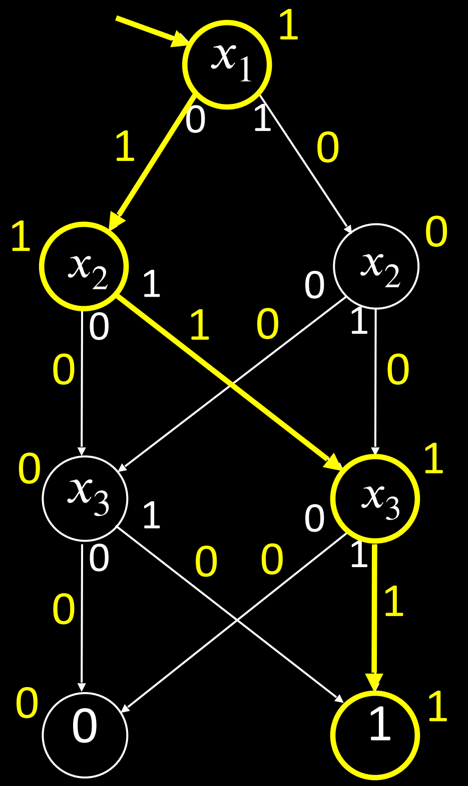
Show by example: Input is $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$

The BP follows its execution path.

Label all nodes and edges on the execution path with 1 and off the execution path with 0.

Boolean Labeling

Alternative way to view BP computation



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

The BP follows its execution path.

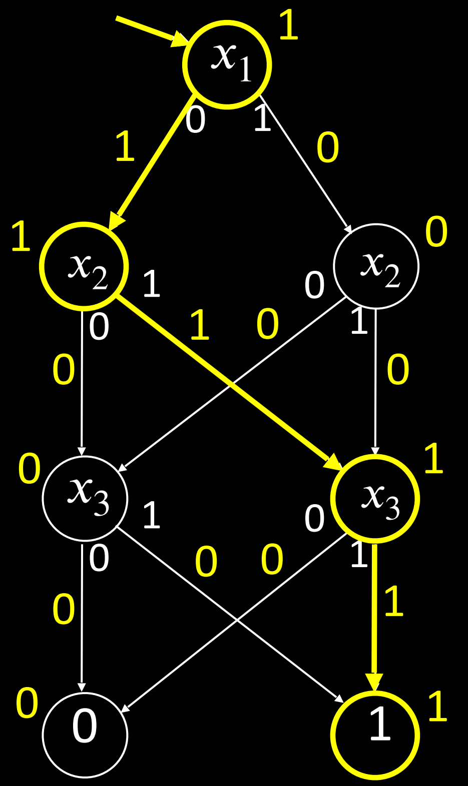
Label all nodes and edges on the execution path with 1

and off the execution path with 0.

Output the label of the output node 1.

Boolean Labeling

Alternative way to view BP computation



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

The BP follows its execution path.

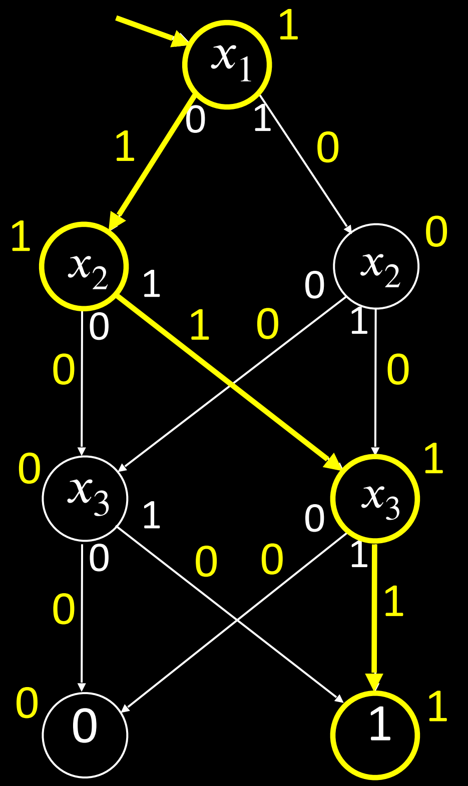
Label all nodes and edges on the execution path with 1 and off the execution path with 0.

Output the label of the output node 1.

Obtain the labeling inductively by using these rules:

Boolean Labeling

Alternative way to view BP computation



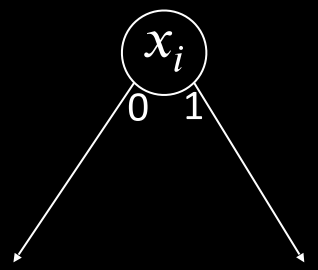
Show by example: Input is $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$

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.

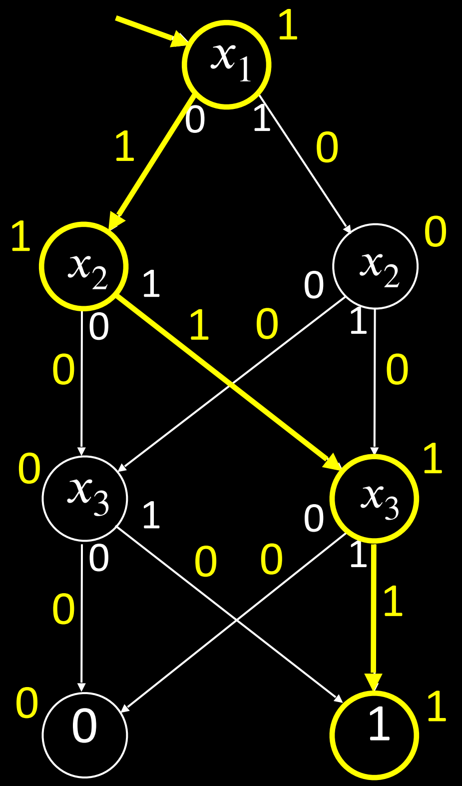
Obtain the labeling inductively by using these rules:



Label edges from nodes

Boolean Labeling

Alternative way to view BP computation



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

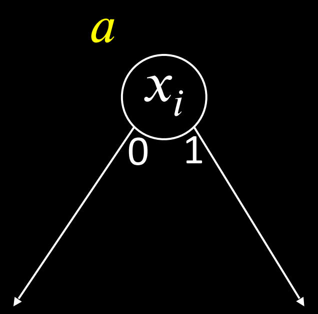
The BP follows its execution path.

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.

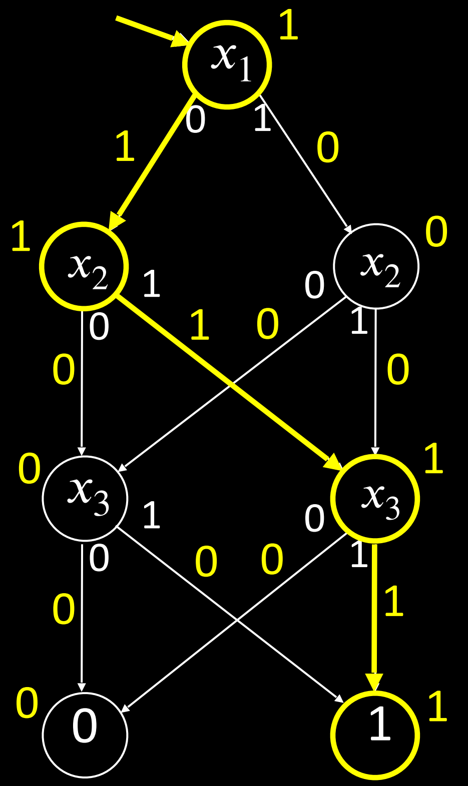
Obtain the labeling inductively by using these rules:



Label edges from nodes

Boolean Labeling

Alternative way to view BP computation



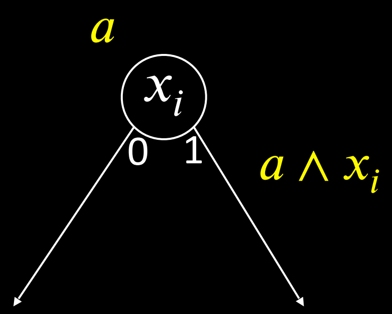
Show by example: Input is $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$

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.

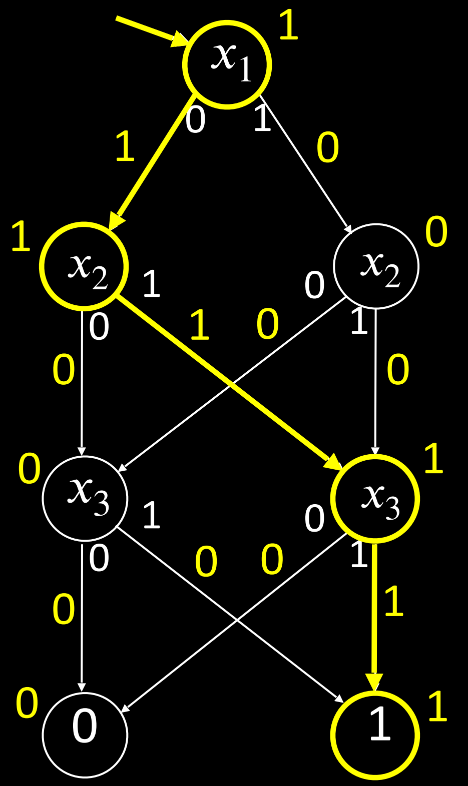
Obtain the labeling inductively by using these rules:



Label edges from nodes

Boolean Labeling

Alternative way to view BP computation



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

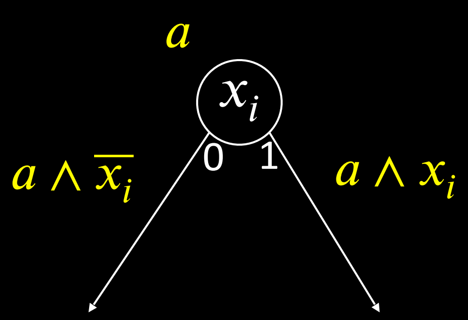
The BP follows its execution path.

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.

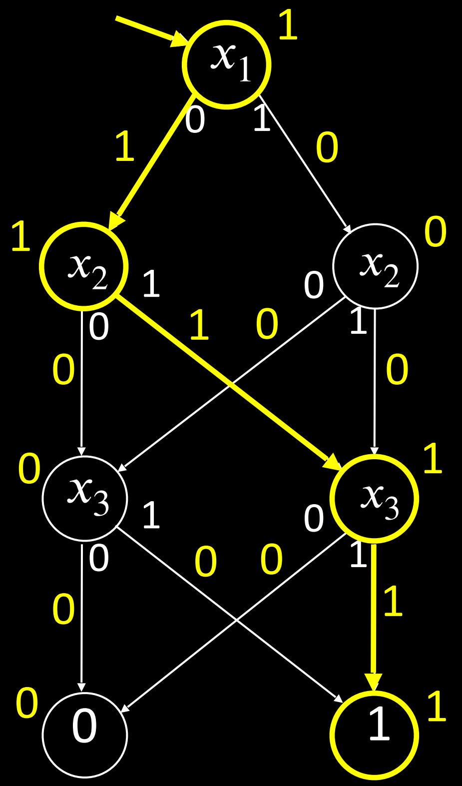
Obtain the labeling inductively by using these rules:



Label edges from nodes

Boolean Labeling

Alternative way to view BP computation



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

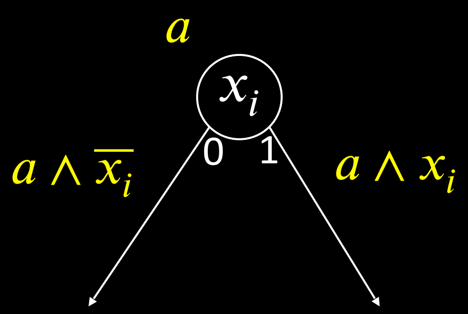
The BP follows its execution path.

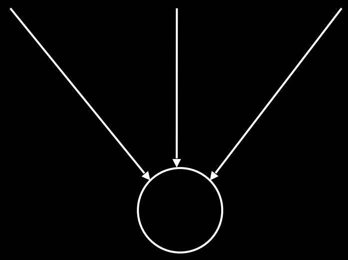
Label all nodes and edges on the execution path with 1

and off the execution path with 0.

Output the label of the output node 1.

Obtain the labeling inductively by using these rules:



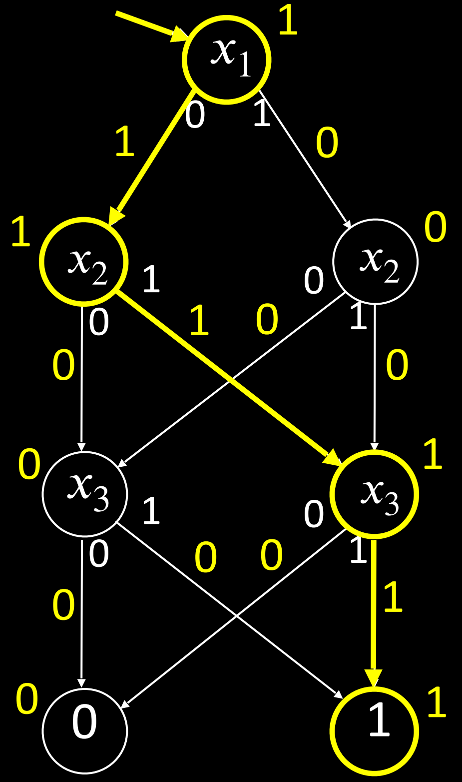


Label edges from nodes

Label nodes from incoming edges

Boolean Labeling

Alternative way to view BP computation



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

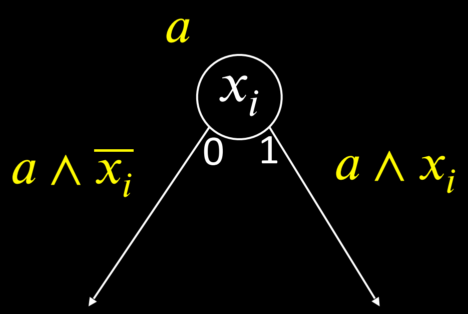
The BP follows its execution path.

Label all nodes and edges on the execution path with 1

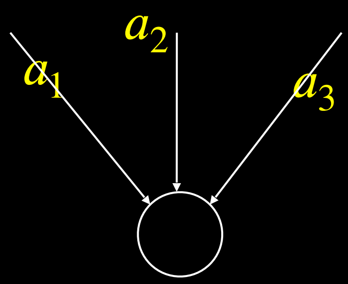
and off the execution path with 0.

Output the label of the output node 1.

Obtain the labeling inductively by using these rules:



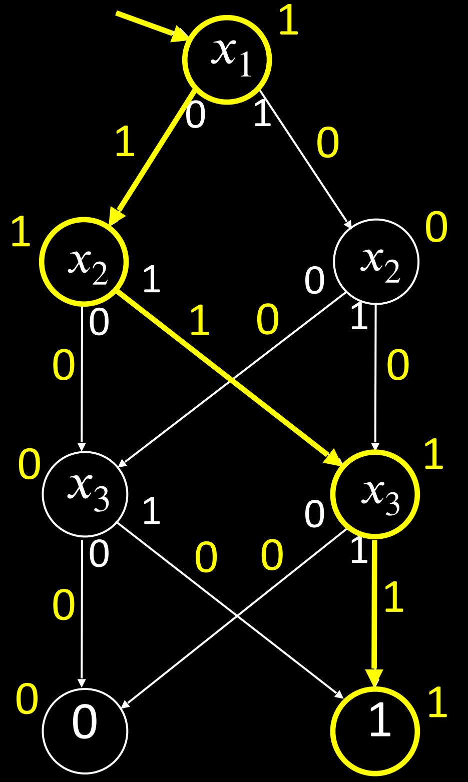
Label edges from nodes



Label nodes from incoming edges

Boolean Labeling

Alternative way to view BP computation



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

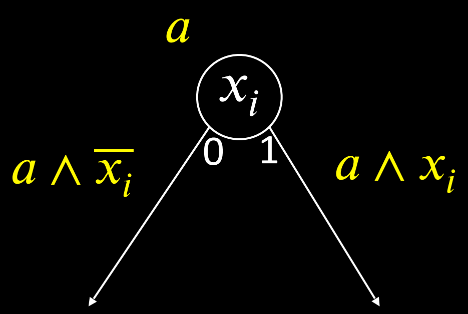
The BP follows its execution path.

Label all nodes and edges on the execution path with 1

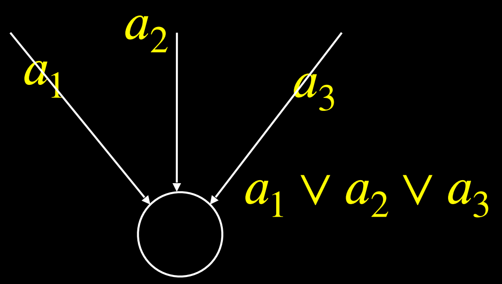
and off the execution path with 0.

Output the label of the output node 1.

Obtain the labeling inductively by using these rules:



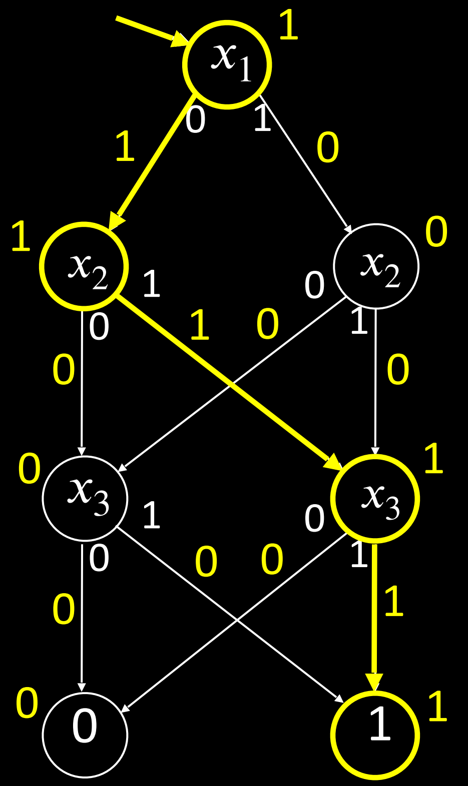
Label edges from nodes



Label nodes from incoming edges

Boolean Labeling

Alternative way to view BP computation



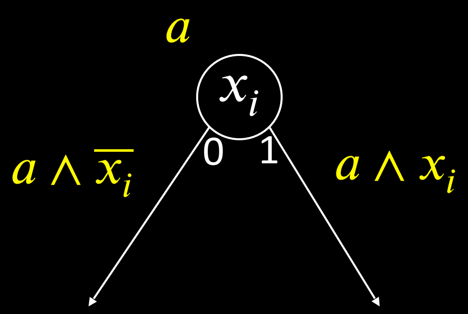
Show by example: Input is $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$

The BP follows its execution path.

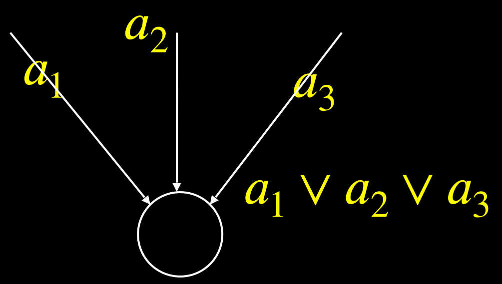
Label all nodes and edges on the execution path with 1 and off the execution path with 0.

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 $\times$.

Arithmetization Method

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

$$a \wedge b \quad \rightarrow \quad a \times b = ab$$

Arithmetization Method

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

$$a \wedge b \quad \to \quad a \times b = ab$$

$$\overline{a} \quad \to \quad (1 - a)$$

Arithmetization Method

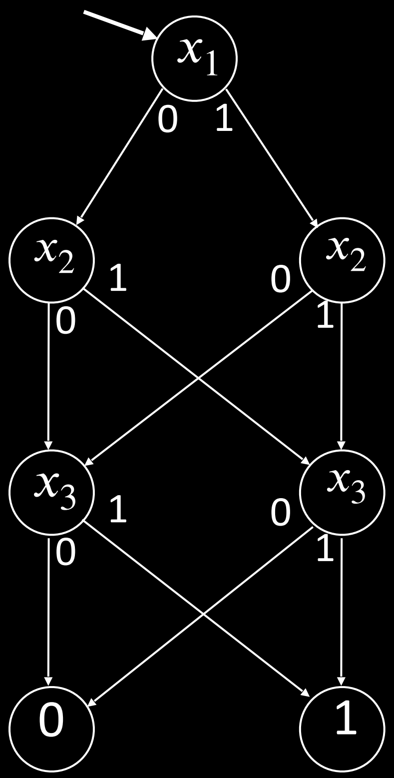
Method: Simulate $\wedge$ and $\vee$ with $+$ and $\times$.

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

Arithmetization Method

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

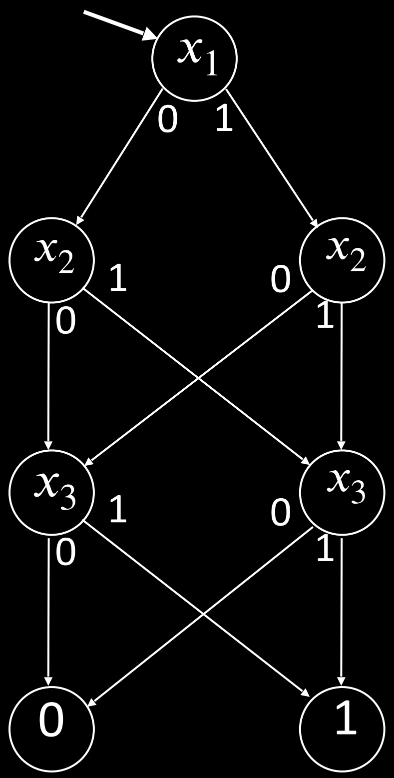
$$a \wedge b \quad \rightarrow \quad a \times b = ab$$
$$\overline{a} \quad \rightarrow \quad (1 - a)$$
$$a \vee b \quad \rightarrow \quad a + b - ab$$



Arithmetization Method

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

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

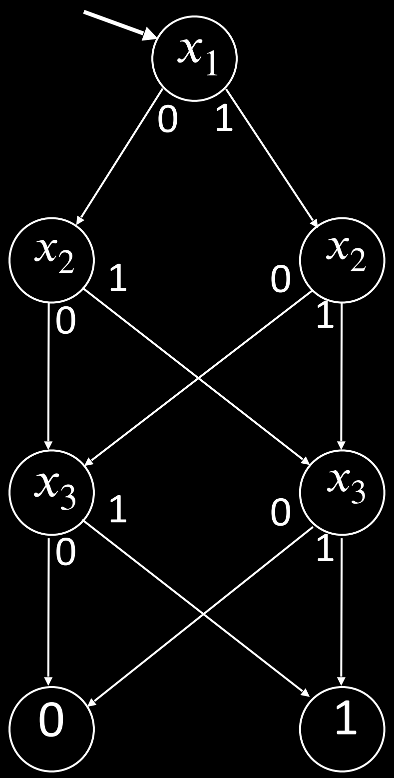


Replace Boolean labeling with arithmetical labeling

Arithmetization Method

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

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



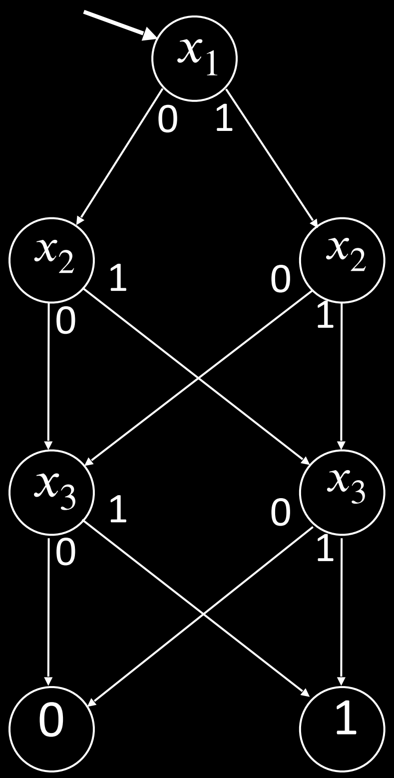
Replace Boolean labeling with arithmetical labeling

Inductive rules:

Arithmetization Method

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

$$
\begin{aligned}
a \wedge b \quad &\rightarrow \quad a \times b = ab \\
\overline{a} \quad &\rightarrow \quad (1 - a) \\
a \vee b \quad &\rightarrow \quad a + b - ab
\end{aligned}
$$



Replace Boolean labeling with arithmetical labeling

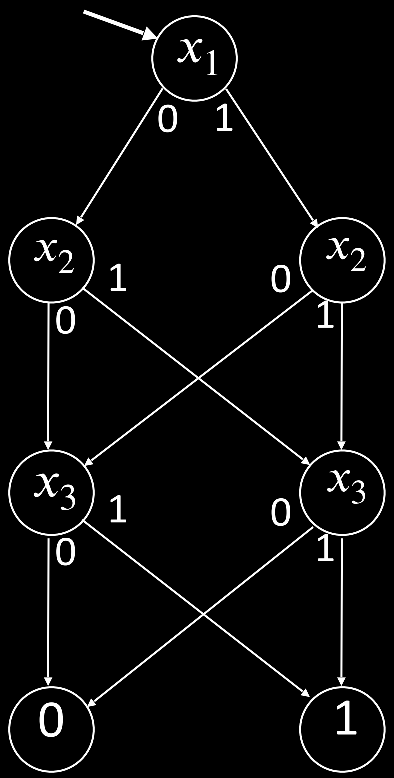
Inductive rules:

Start node labeled 1

Arithmetization Method

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

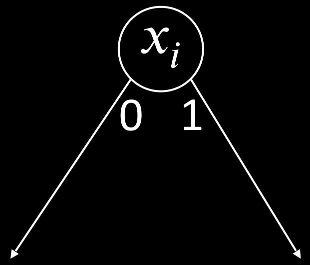
$$a \wedge b \quad \to \quad a \times b = ab$$
$$\overline{a} \quad \to \quad (1 - a)$$
$$a \vee b \quad \to \quad a + b - ab$$



Replace Boolean labeling with arithmetical labeling

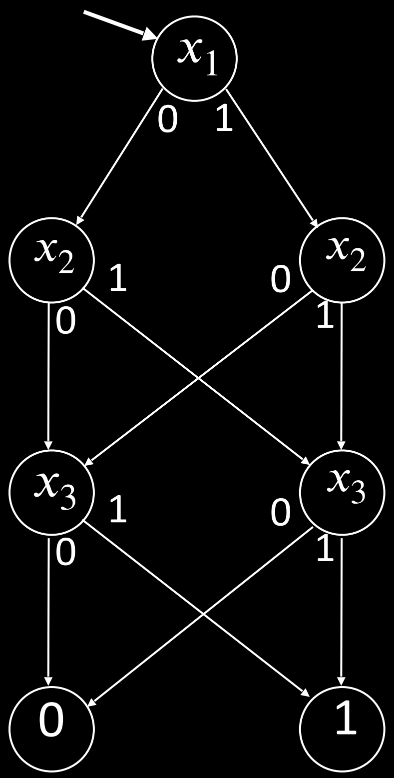
Inductive rules:

Start node labeled 1



Arithmetization Method

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

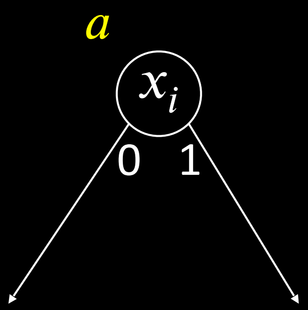


$$a \wedge b \;\rightarrow\; a \times b = ab$$
$$\overline{a} \;\rightarrow\; (1 - a)$$
$$a \vee b \;\rightarrow\; a + b - ab$$

Replace Boolean labeling with arithmetical labeling

Inductive rules:

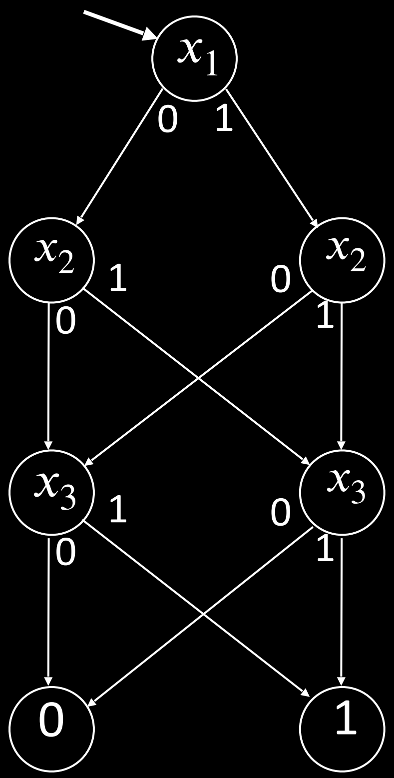
Start node labeled 1



Arithmetization Method

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

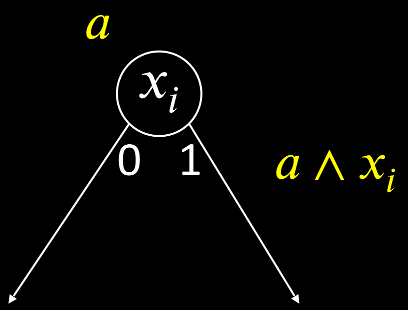
$$
\begin{aligned}
a \wedge b &\rightarrow a \times b = ab \\
\overline{a} &\rightarrow (1 - a) \\
a \vee b &\rightarrow a + b - ab
\end{aligned}
$$



Replace Boolean labeling with arithmetical labeling

Inductive rules:

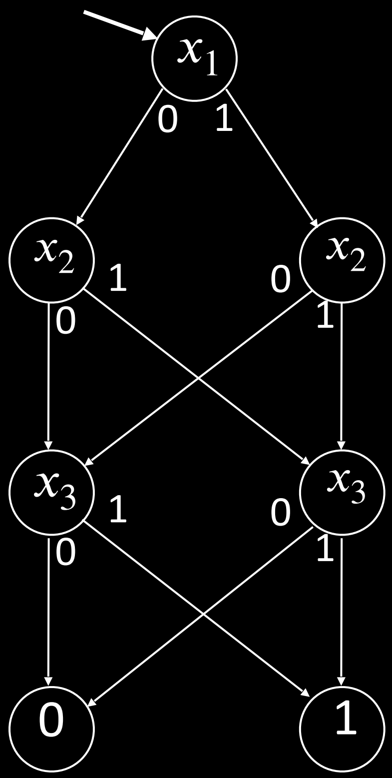
Start node labeled 1



Arithmetization Method

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

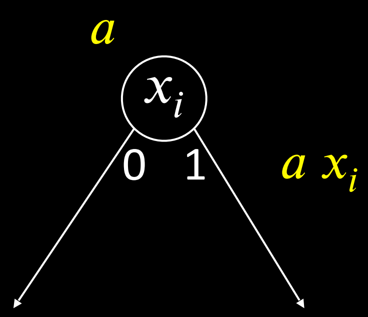
$$a \wedge b \;\rightarrow\; a \times b = ab$$
$$\overline{a} \;\rightarrow\; (1 - a)$$
$$a \vee b \;\rightarrow\; a + b - ab$$



Replace Boolean labeling with arithmetical labeling

Inductive rules:

Start node labeled 1



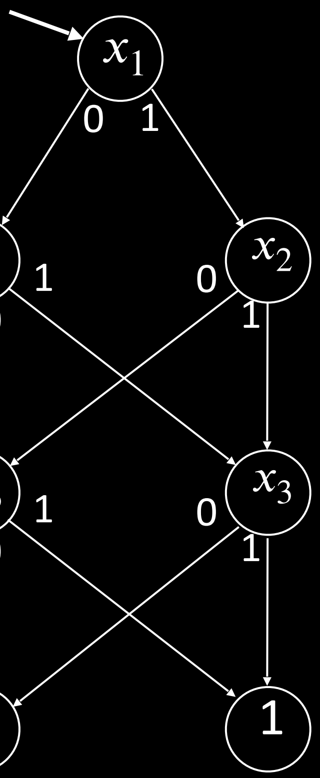
Arithmetization Method

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

$$a \wedge b \quad \rightarrow \quad a \times b = ab$$

$$\overline{a} \quad \rightarrow \quad (1 - a)$$

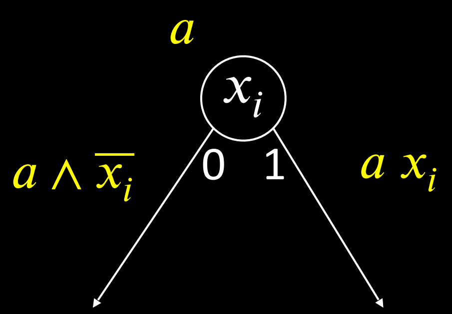
$$a \vee b \quad \rightarrow \quad a + b - ab$$



Replace Boolean labeling with arithmetical labeling

Inductive rules:

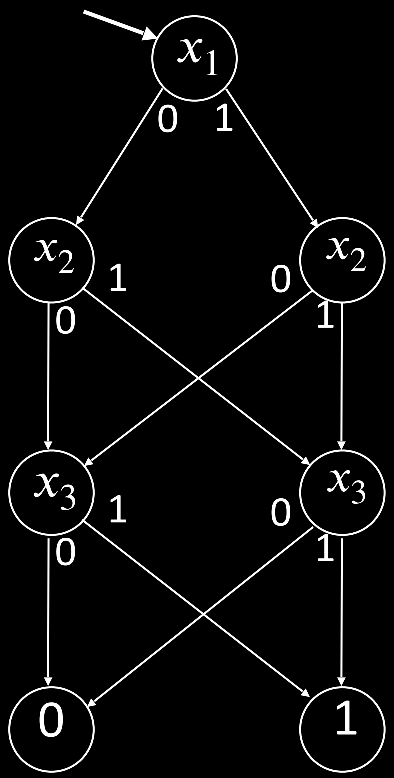
Start node labeled 1



Arithmetization Method

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

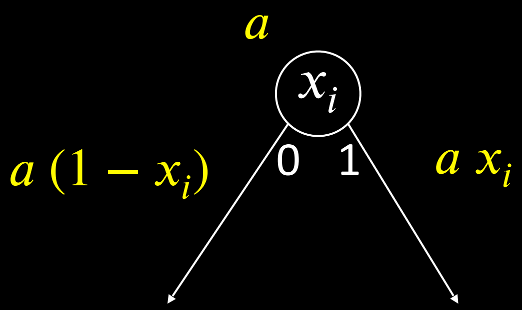
$$a \wedge b \;\rightarrow\; a \times b = ab$$
$$\overline{a} \;\rightarrow\; (1-a)$$
$$a \vee b \;\rightarrow\; a + b - ab$$



Replace Boolean labeling with arithmetical labeling

Inductive rules:

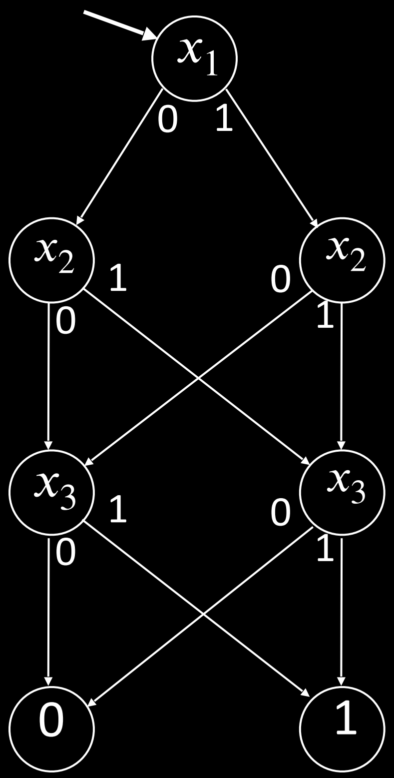
Start node labeled 1



Arithmetization Method

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

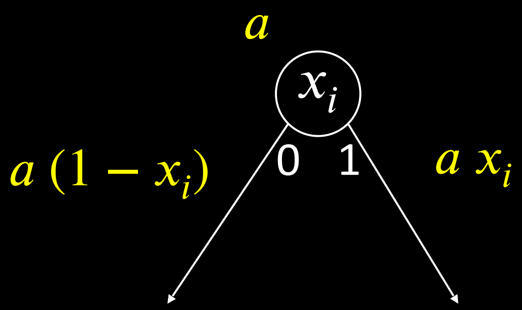
$$
\begin{aligned}
a \wedge b \quad &\rightarrow \quad a \times b = ab \\
\overline{a} \quad &\rightarrow \quad (1 - a) \\
a \vee b \quad &\rightarrow \quad a + b - ab
\end{aligned}
$$

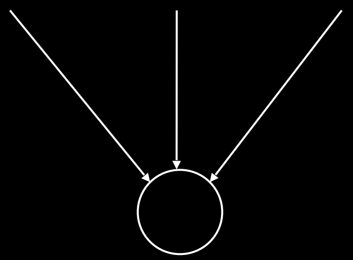


Replace Boolean labeling with arithmetical labeling

Inductive rules:

Start node labeled 1

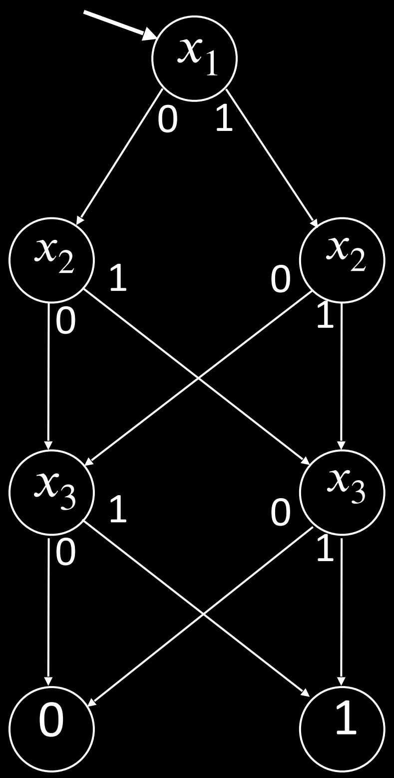




Arithmetization Method

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

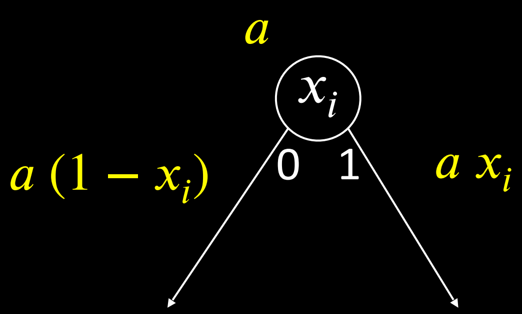
$$a \wedge b \quad \rightarrow \quad a \times b = ab$$
$$\bar{a} \quad \rightarrow \quad (1 - a)$$
$$a \vee b \quad \rightarrow \quad a + b - ab$$

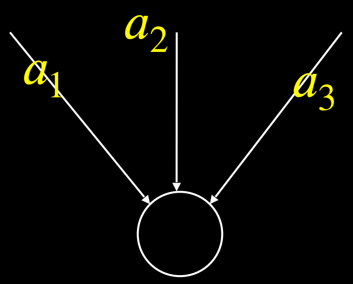


Replace Boolean labeling with arithmetical labeling

Inductive rules:

Start node labeled 1

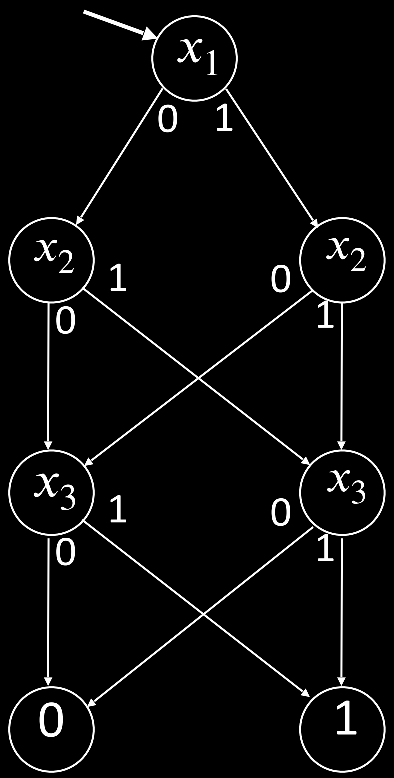




Arithmetization Method

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

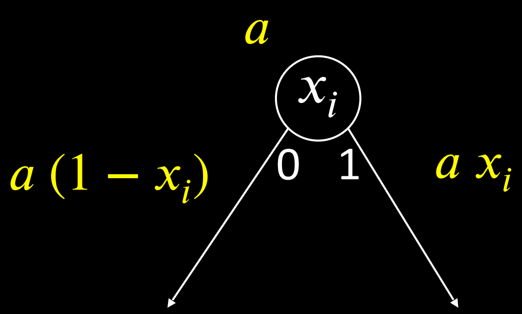
$$a \wedge b \quad \to \quad a \times b = ab$$
$$\overline{a} \quad \to \quad (1 - a)$$
$$a \vee b \quad \to \quad a + b - ab$$

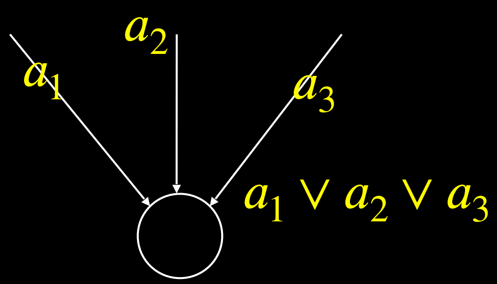


Replace Boolean labeling with arithmetical labeling

Inductive rules:

Start node labeled 1

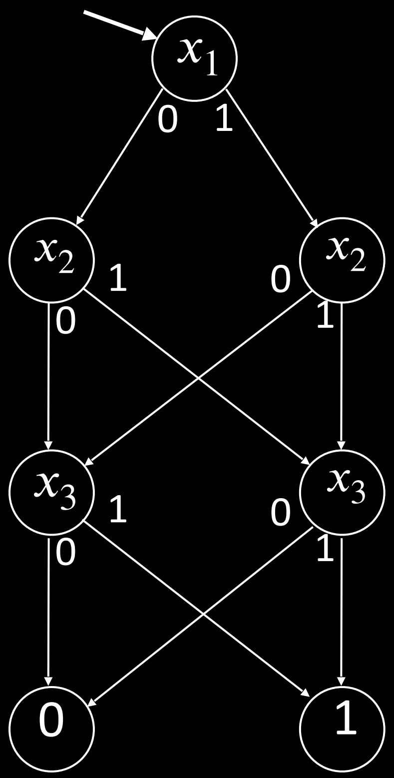




Arithmetization Method

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

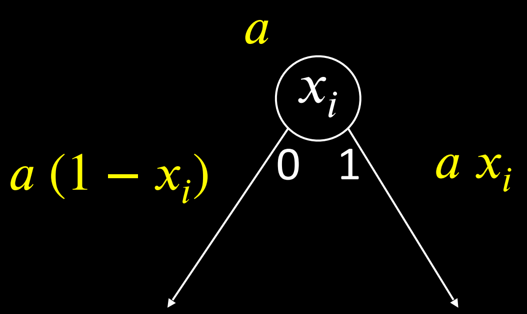
$$a \wedge b \quad \rightarrow \quad a \times b = ab$$
$$\overline{a} \quad \rightarrow \quad (1 - a)$$
$$a \vee b \quad \rightarrow \quad a + b - ab$$

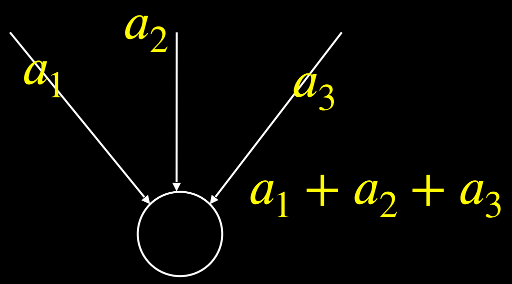


Replace Boolean labeling with arithmetical labeling

Inductive rules:

Start node labeled 1

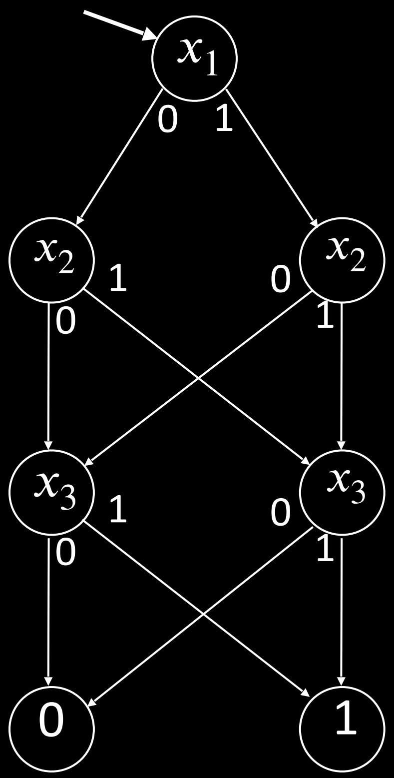




Arithmetization Method

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

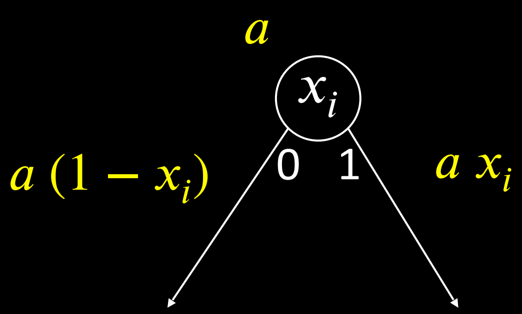
$$a \wedge b \;\to\; a \times b = ab$$
$$\overline{a} \;\to\; (1 - a)$$
$$a \vee b \;\to\; a + b - ab$$

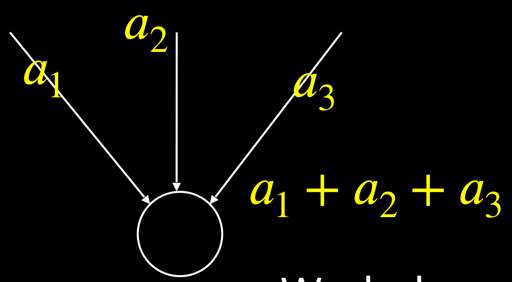


Replace Boolean labeling with arithmetical labeling

Inductive rules:

Start node labeled 1

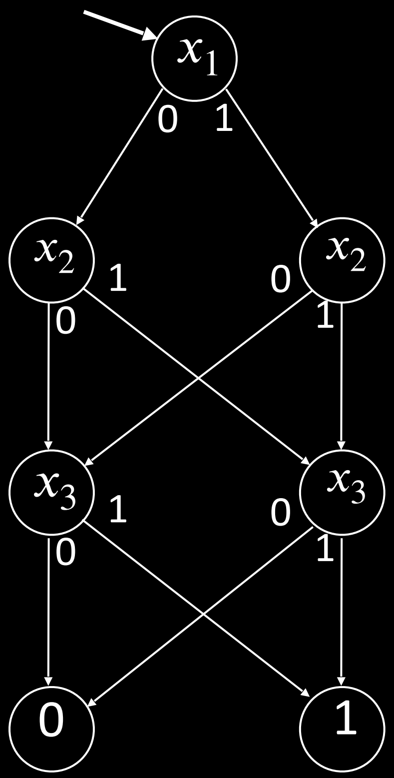




Works because the BP is acyclic.
The execution path can enter a node at most one time.

Arithmetization Method

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

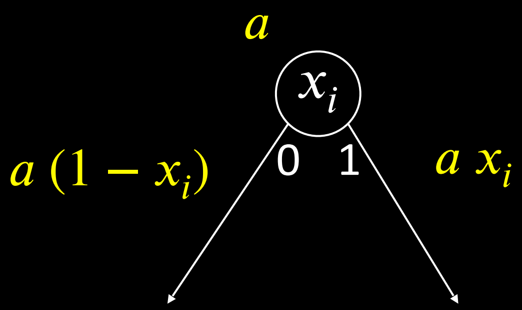


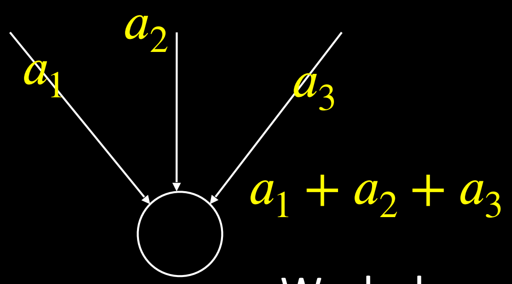
$$
\begin{aligned}
a \wedge b &\rightarrow a \times b = ab \\
\overline{a} &\rightarrow (1 - a) \\
a \vee b &\rightarrow a + b - ab
\end{aligned}
$$

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.

Non-Boolean Inputs

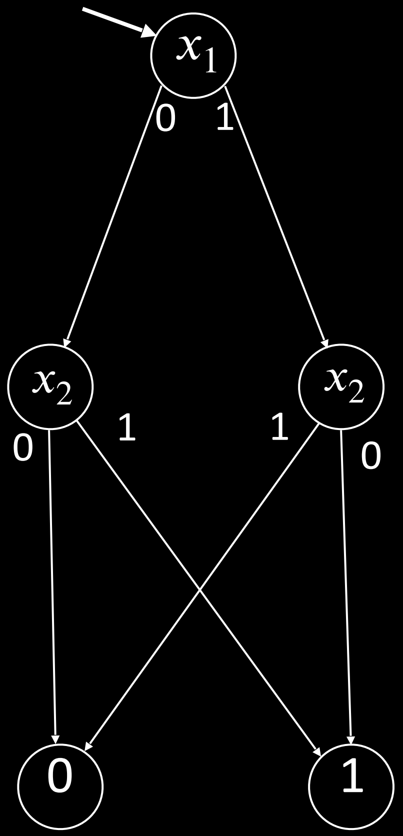
Use the arithmetized interpretation of the BP's computation to define its operation on non-Boolean inputs.

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

Non-Boolean Inputs

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

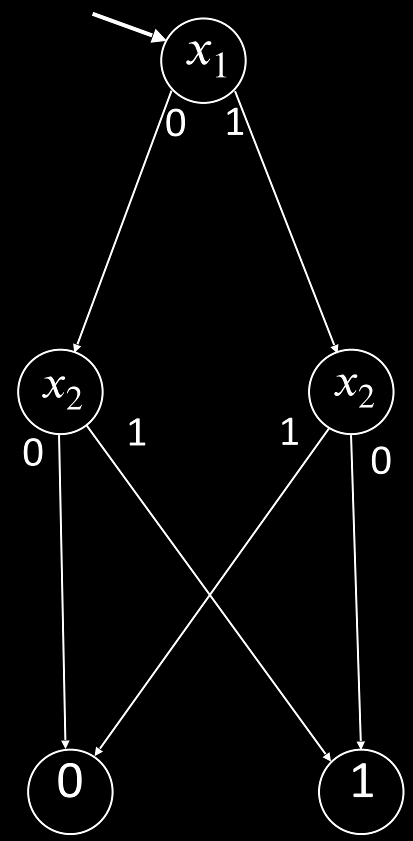
Example: $x_1 = 2, \quad x_2 = 3$



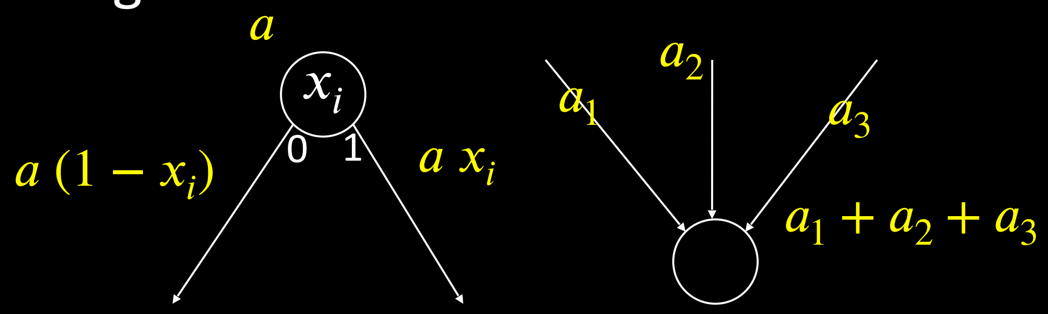
Non-Boolean Inputs

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

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



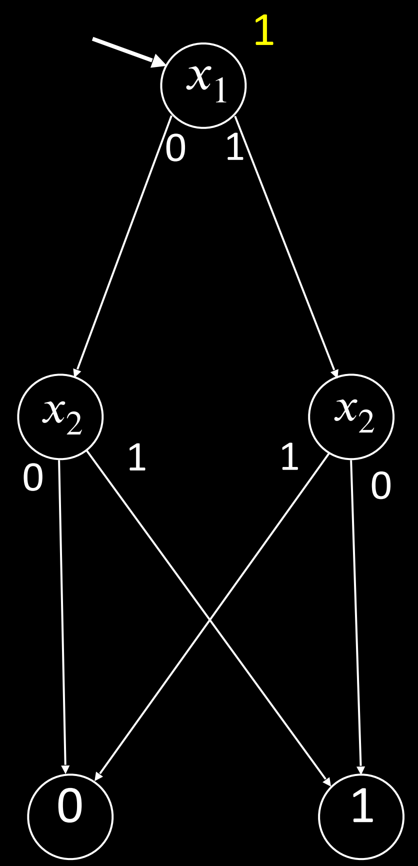
Recall labeling rules:



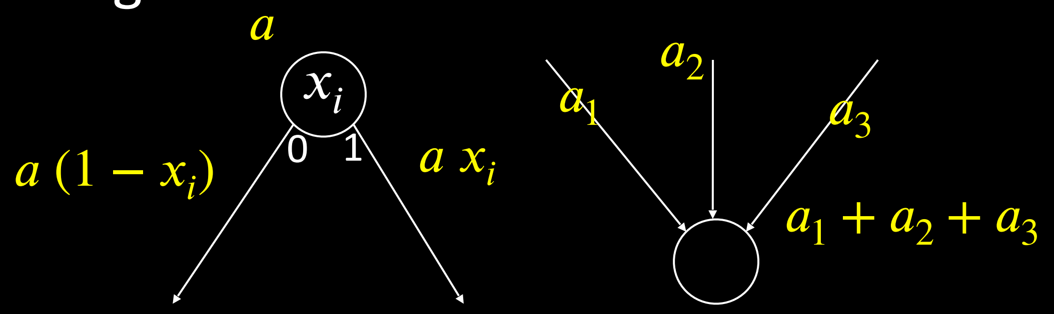
Non-Boolean Inputs

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

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



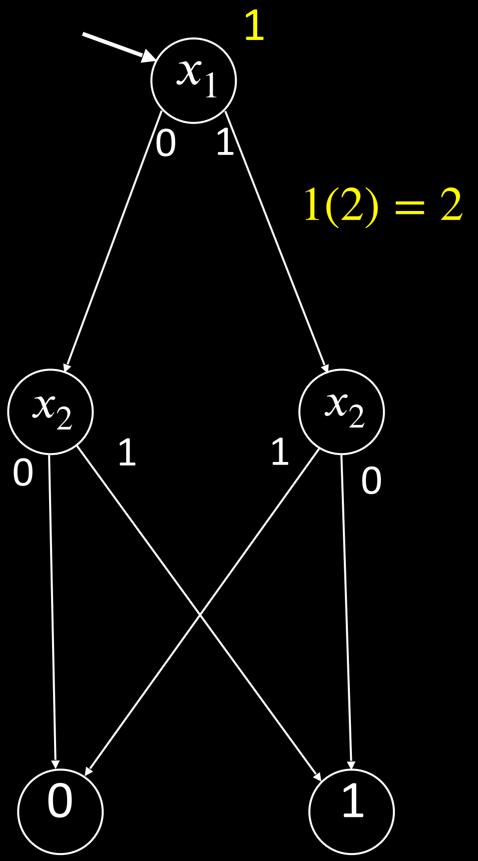
Recall labeling rules:



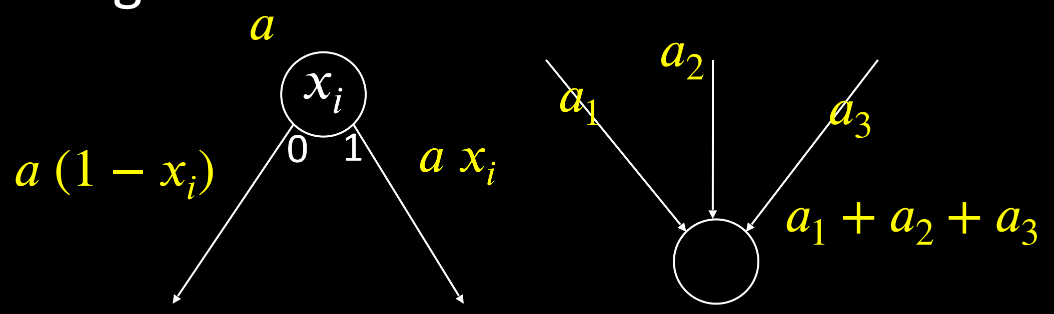
Non-Boolean Inputs

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

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



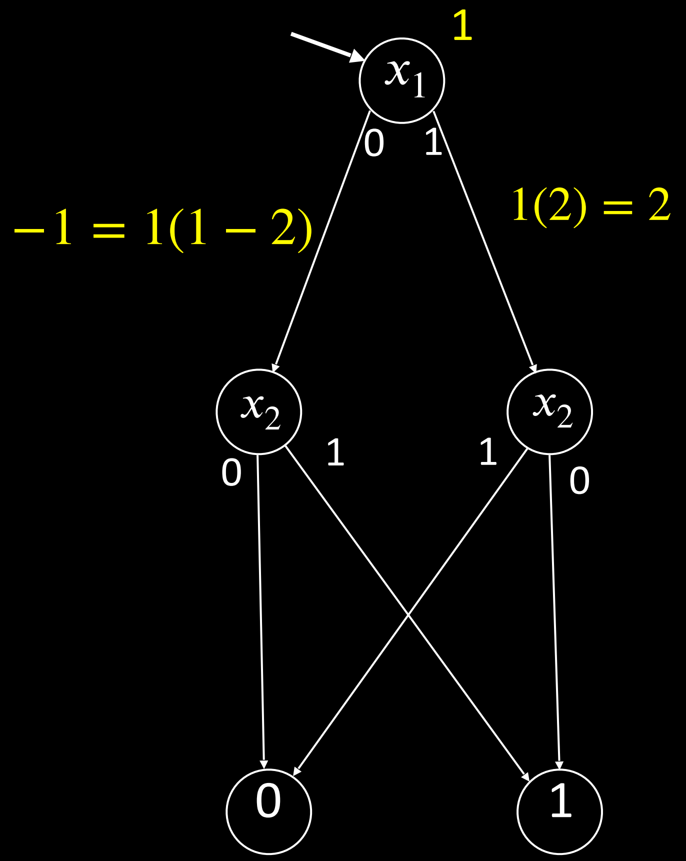
Recall labeling rules:



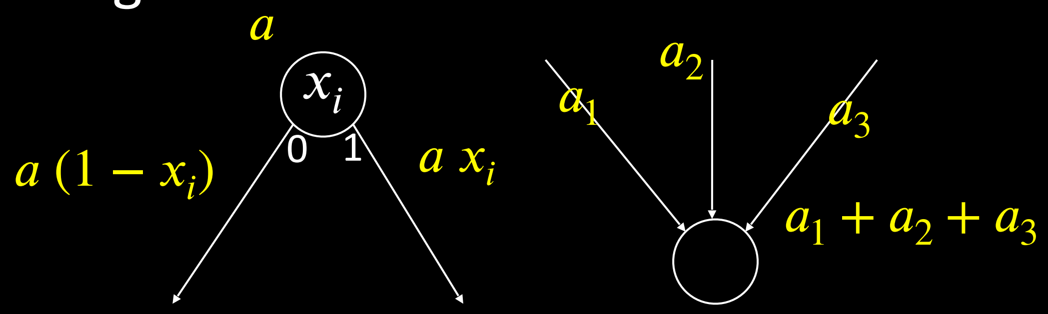
Non-Boolean Inputs

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

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



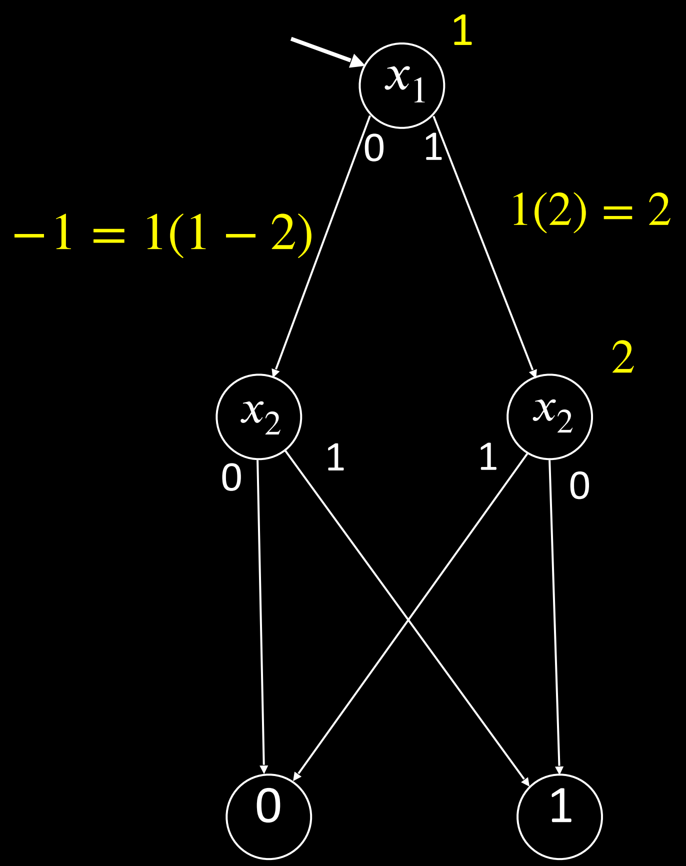
Recall labeling rules:



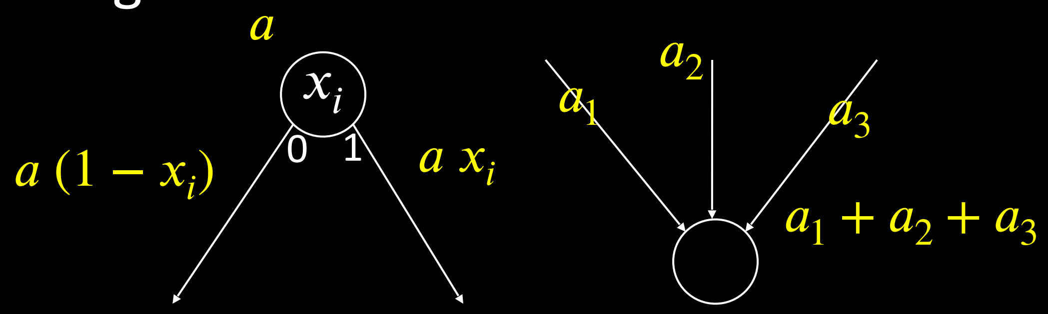
Non-Boolean Inputs

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

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



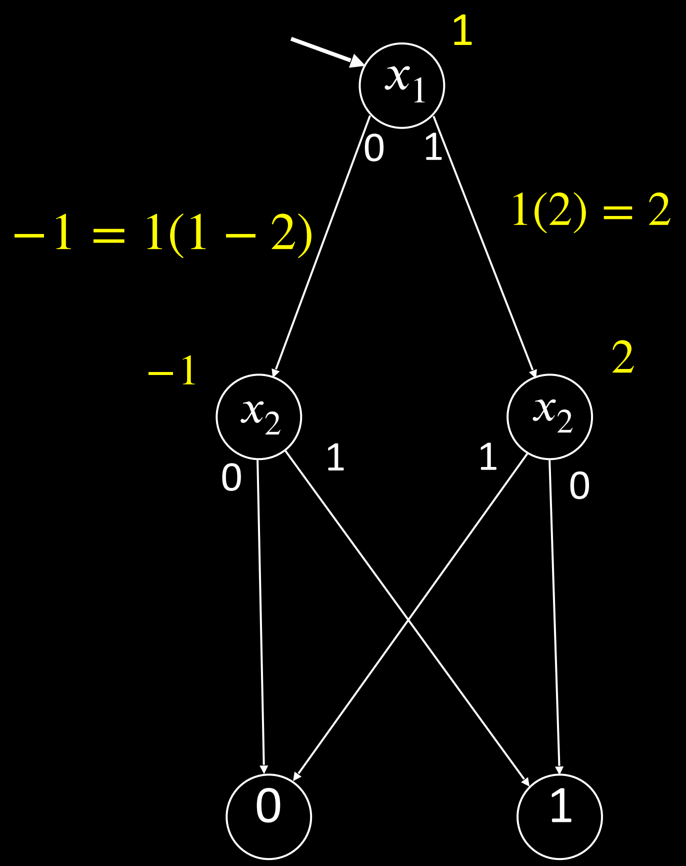
Recall labeling rules:



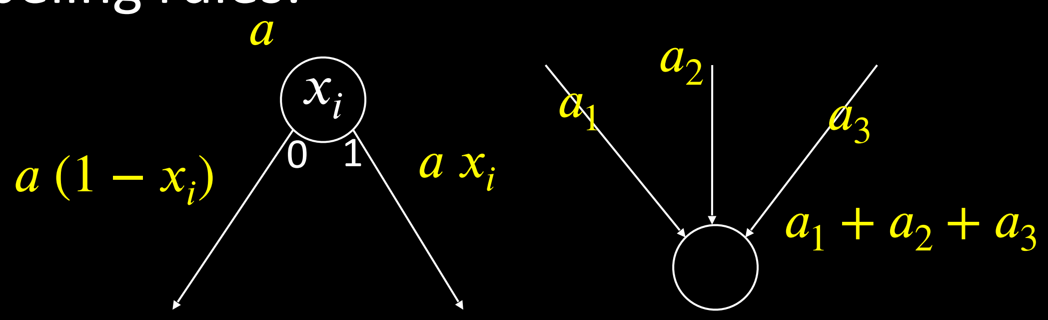
Non-Boolean Inputs

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

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



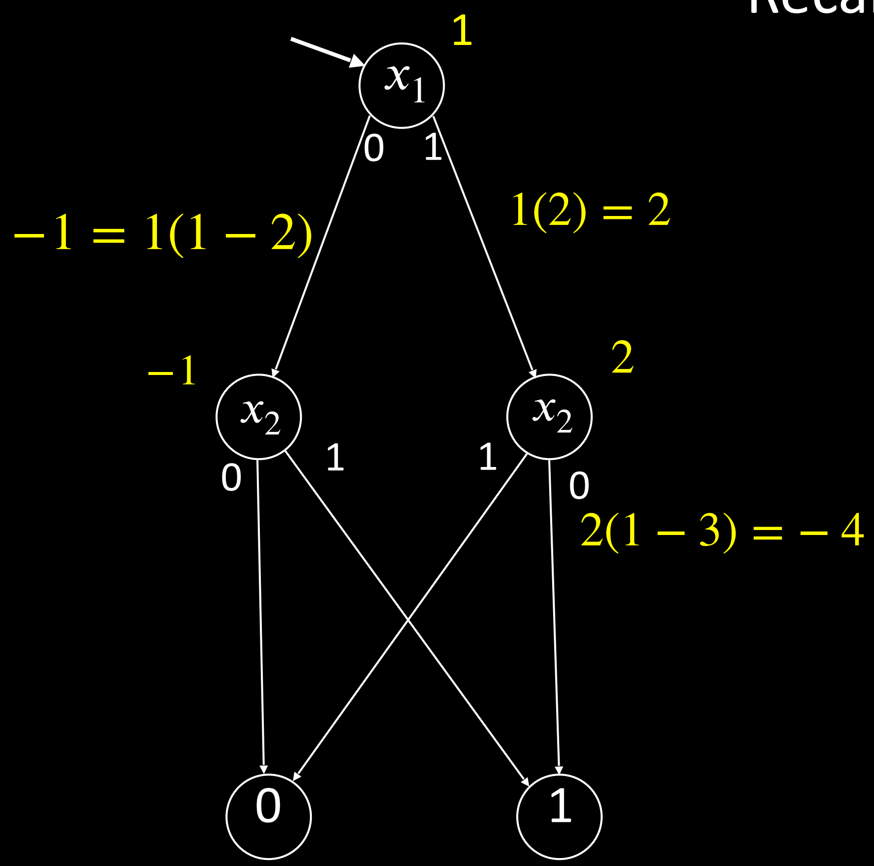
Recall labeling rules:



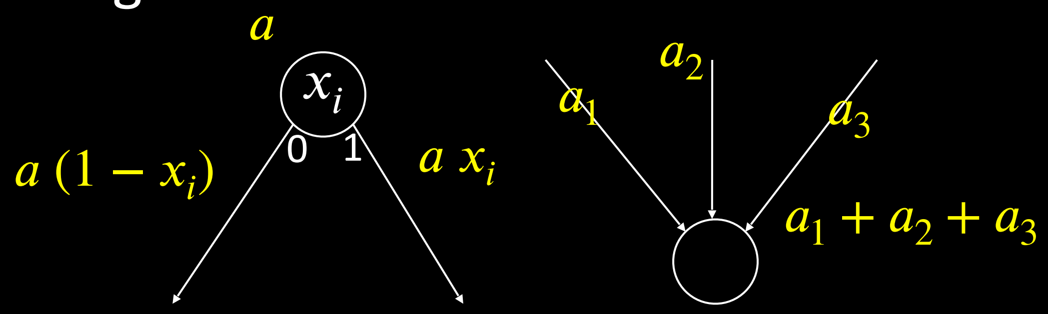
Non-Boolean Inputs

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

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



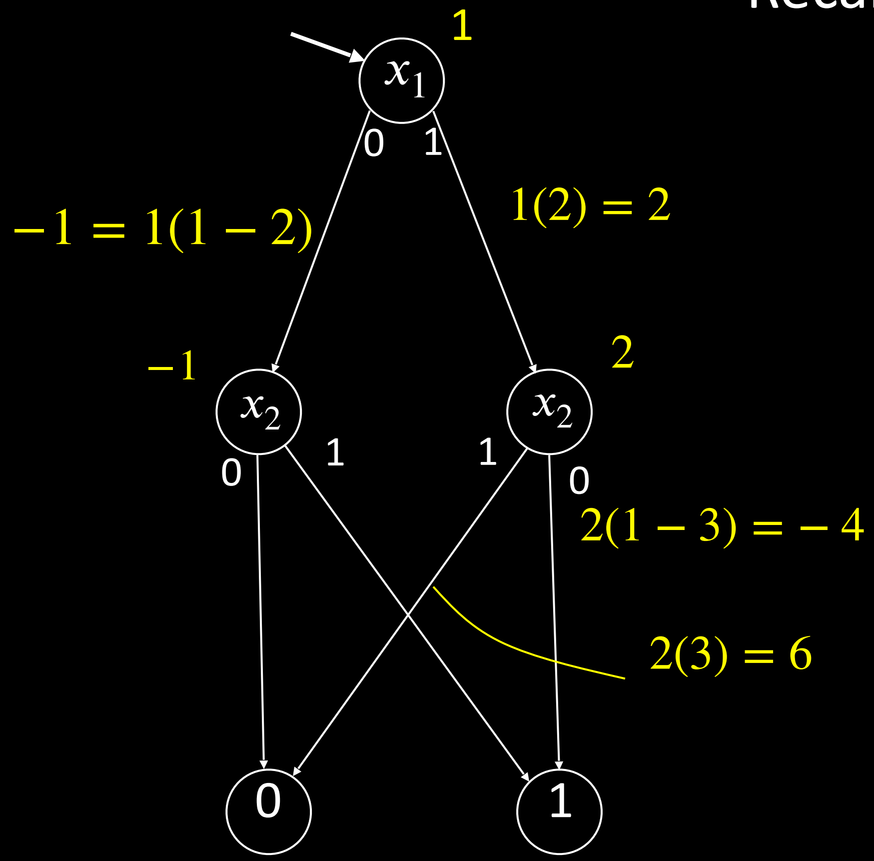
Recall labeling rules:



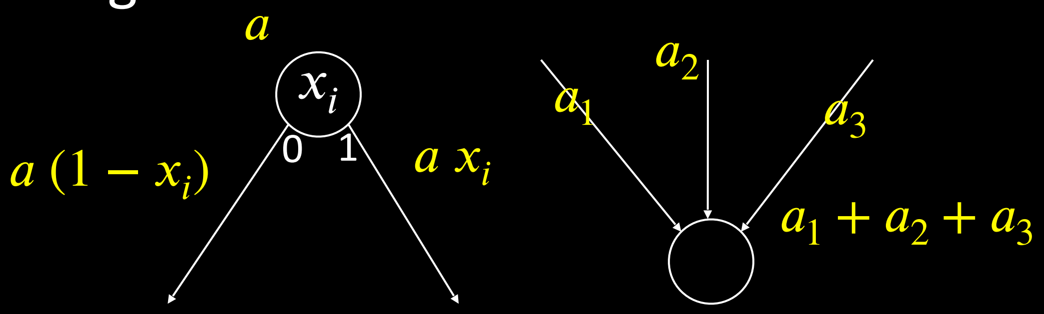
Non-Boolean Inputs

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

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



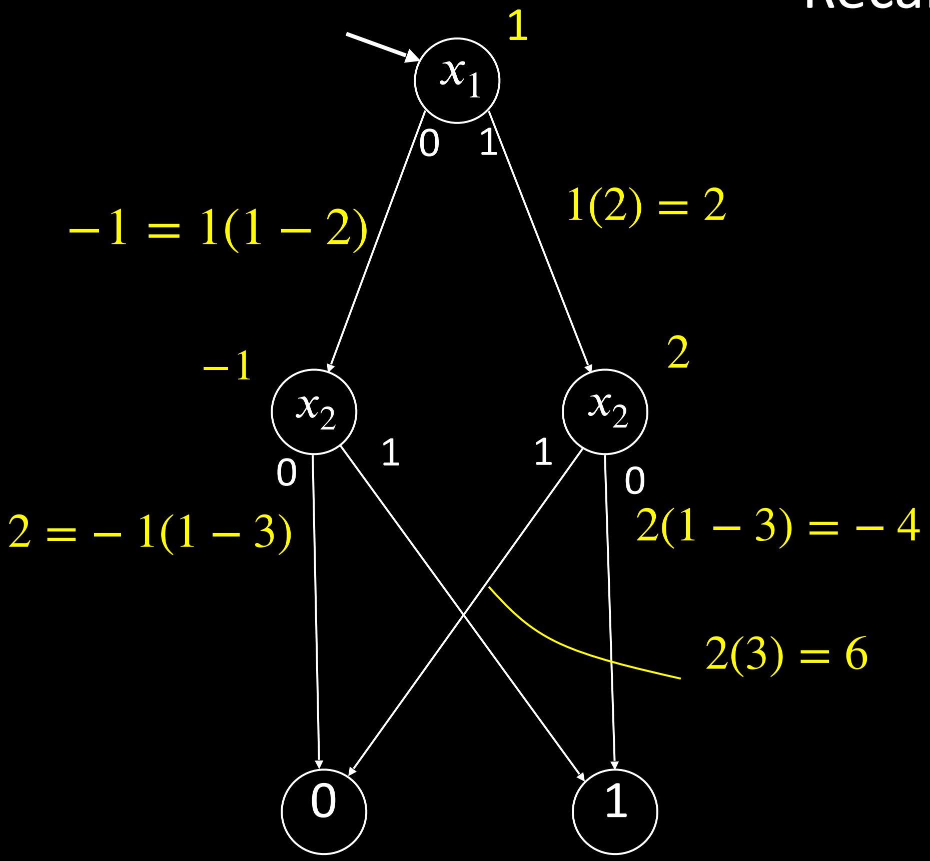
Recall labeling rules:



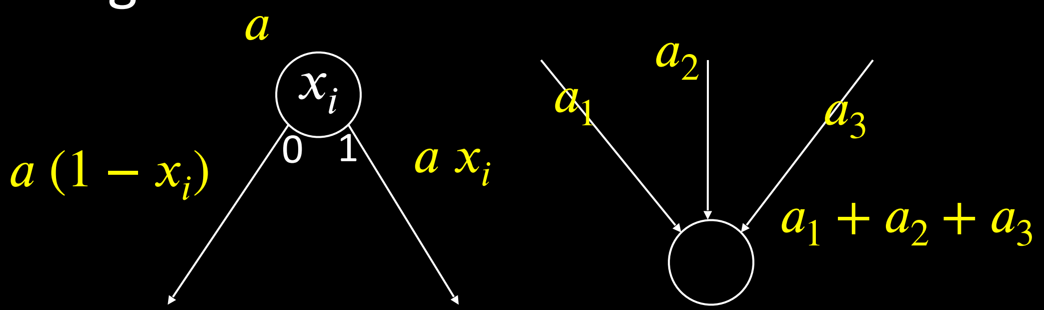
Non-Boolean Inputs

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

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



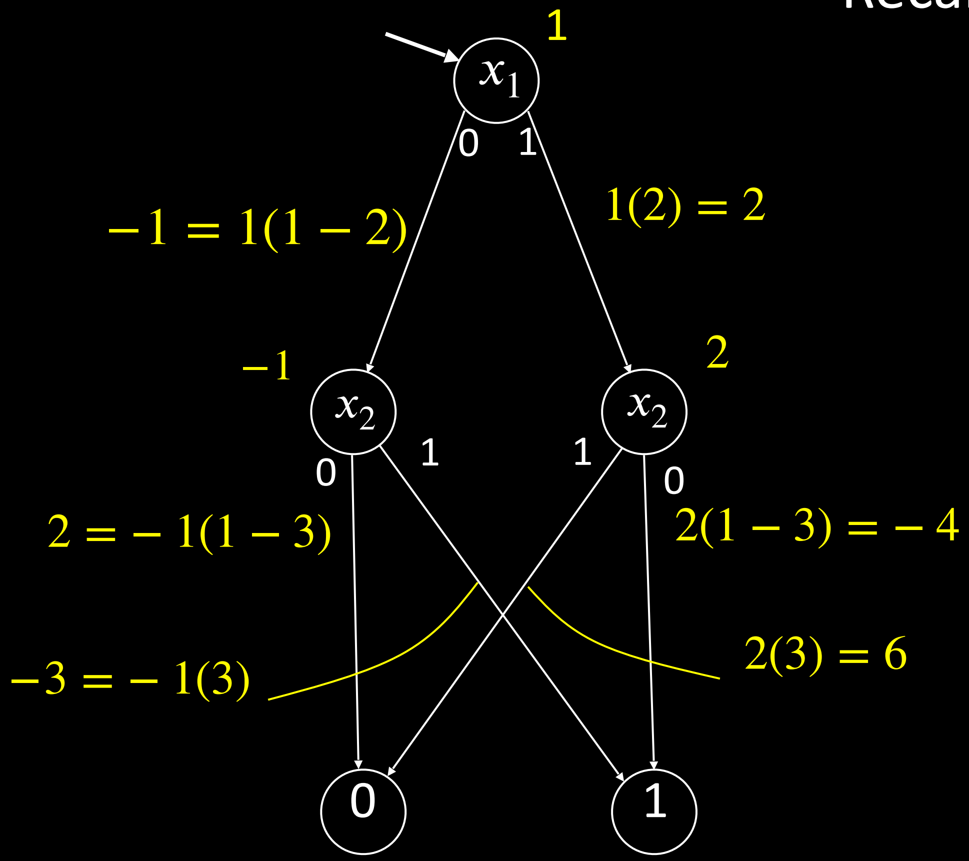
Recall labeling rules:



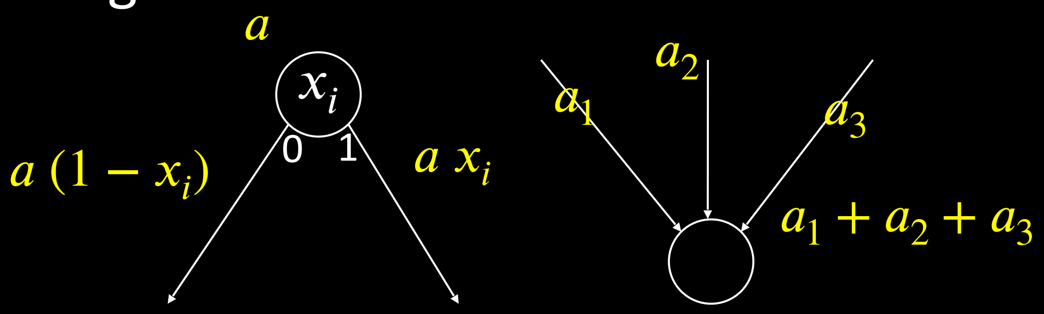
Non-Boolean Inputs

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

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



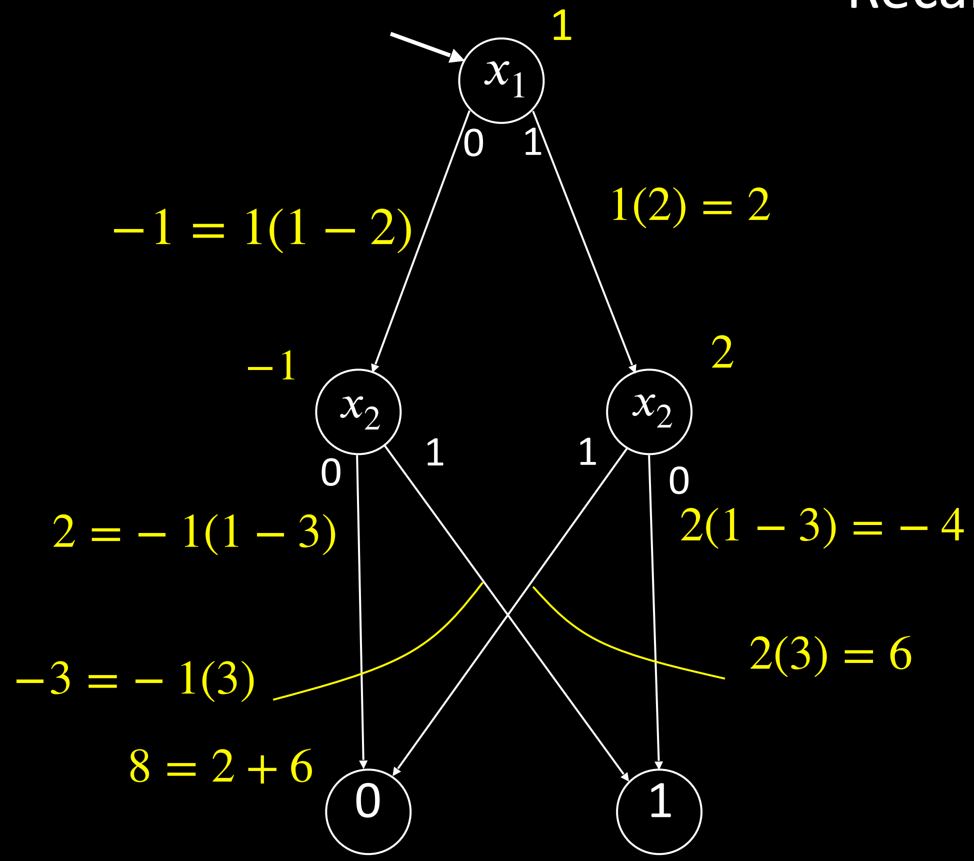
Recall labeling rules:



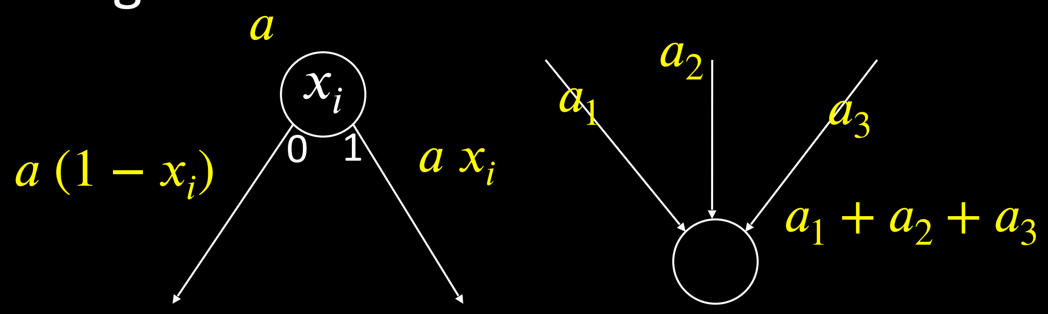
Non-Boolean Inputs

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

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



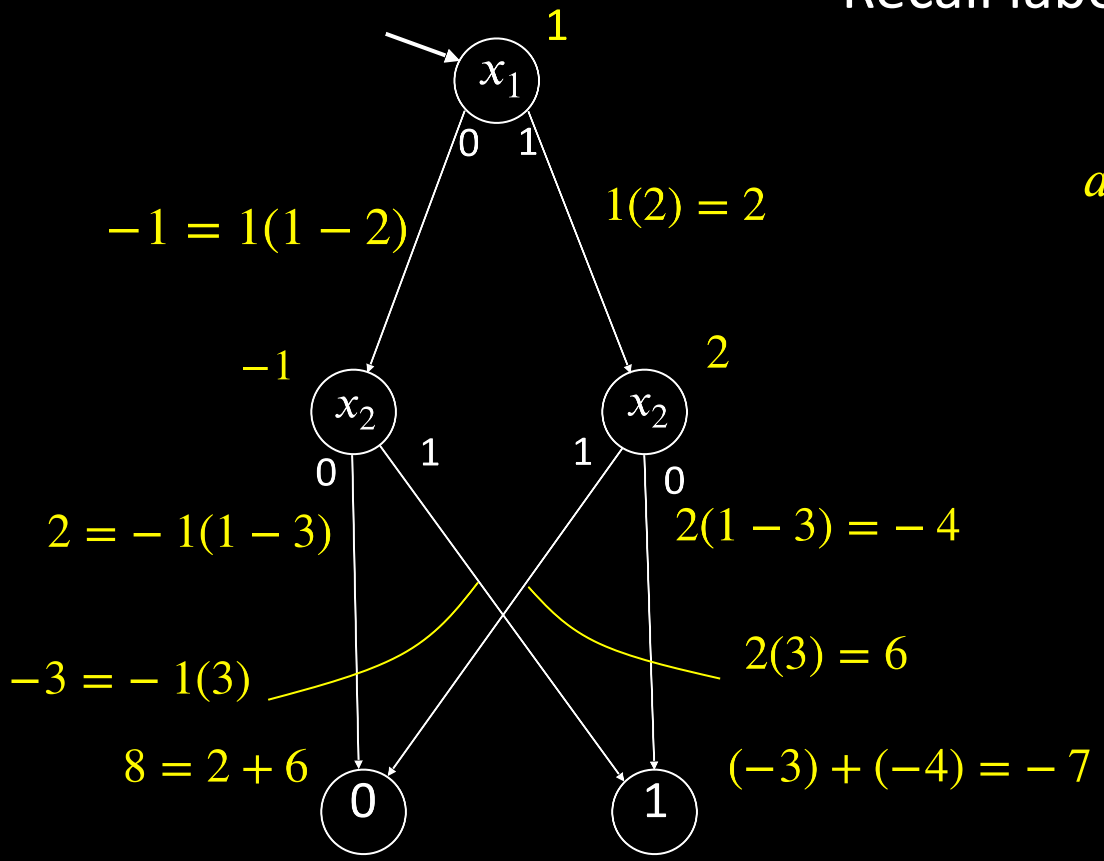
Recall labeling rules:



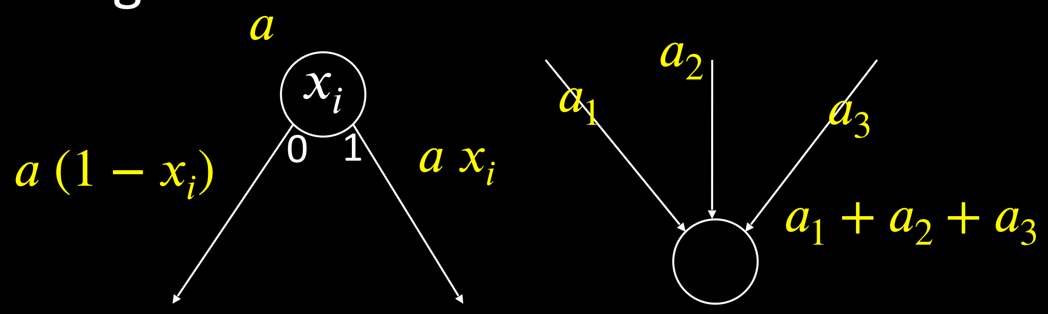
Non-Boolean Inputs

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

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



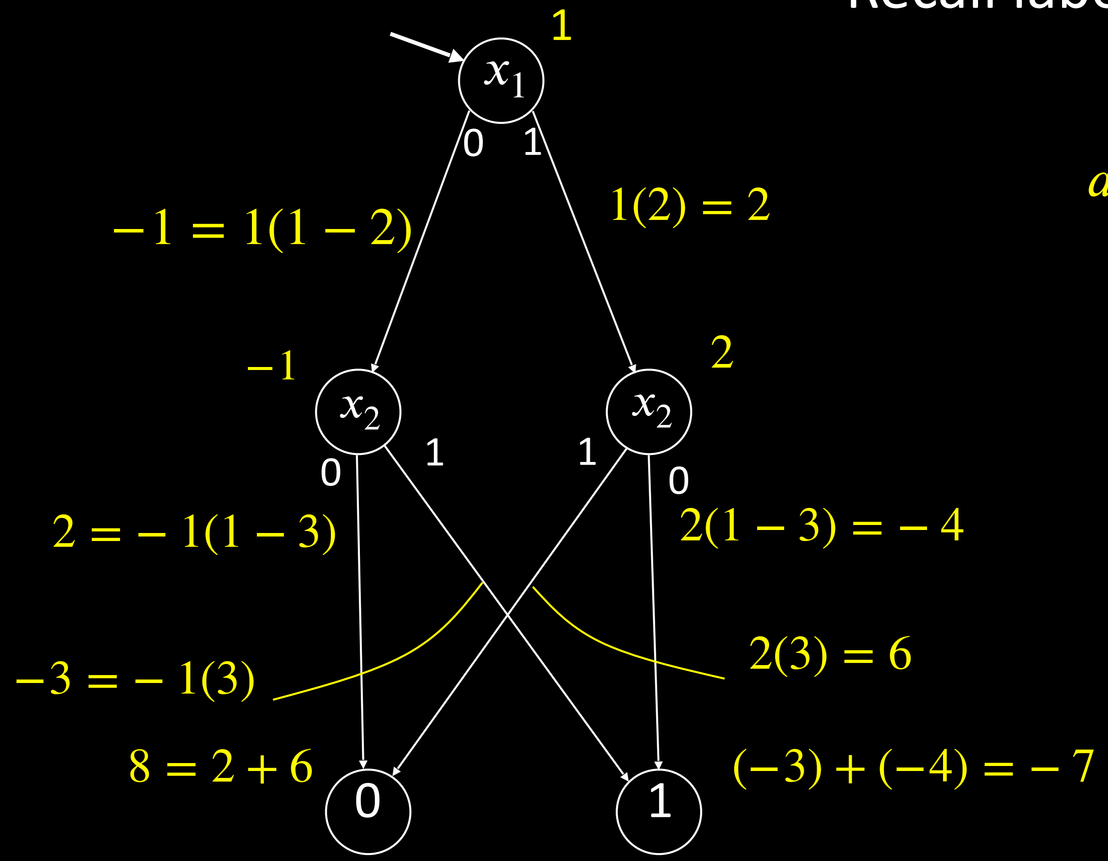
Recall labeling rules:



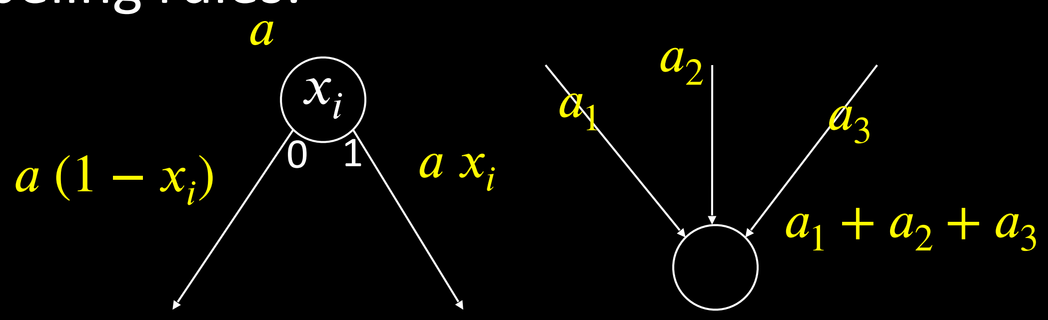
Non-Boolean Inputs

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

Example: $x_1 = 2, \quad x_2 = 3$ Output = -7



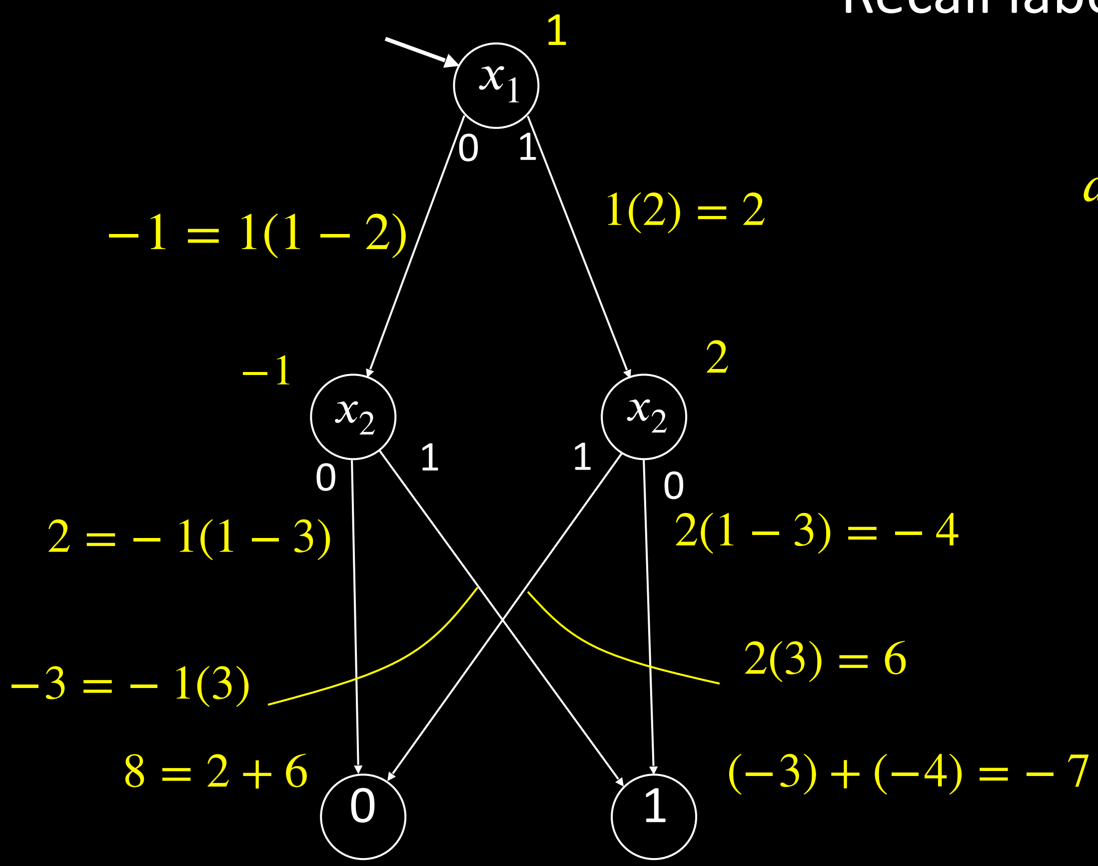
Recall labeling rules:



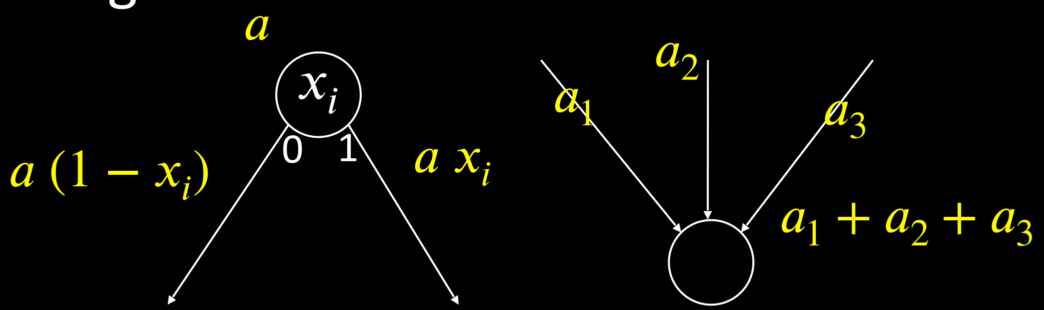
Non-Boolean Inputs

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

Example: $x_1 = 2$, $x_2 = 3$ Output = -7



Recall labeling rules:

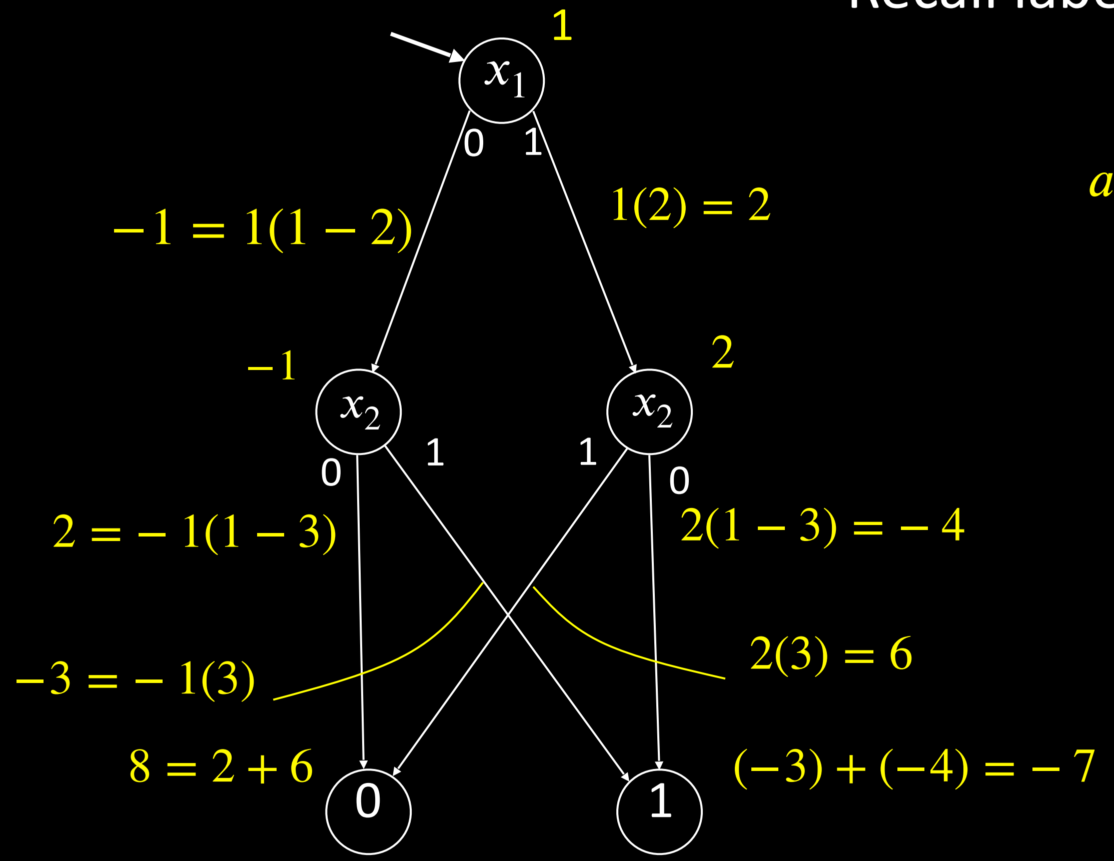


Revised M for EQROBP: "On input $\langle B_1, B_2\rangle$

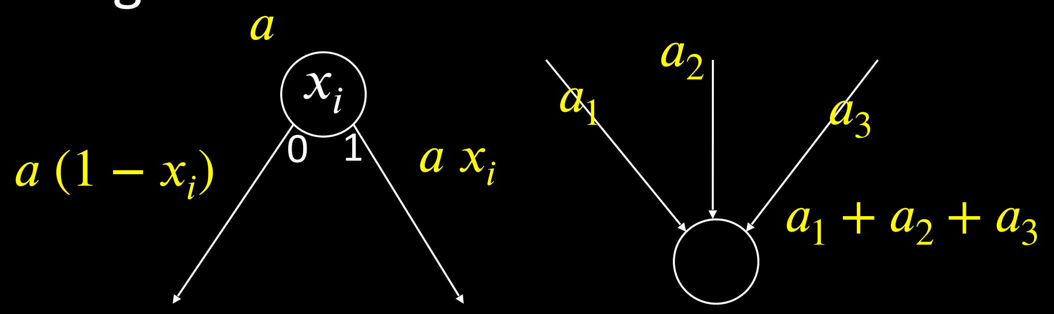
Non-Boolean Inputs

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

Example: $x_1 = 2, \quad x_2 = 3$ Output = -7



Recall labeling rules:



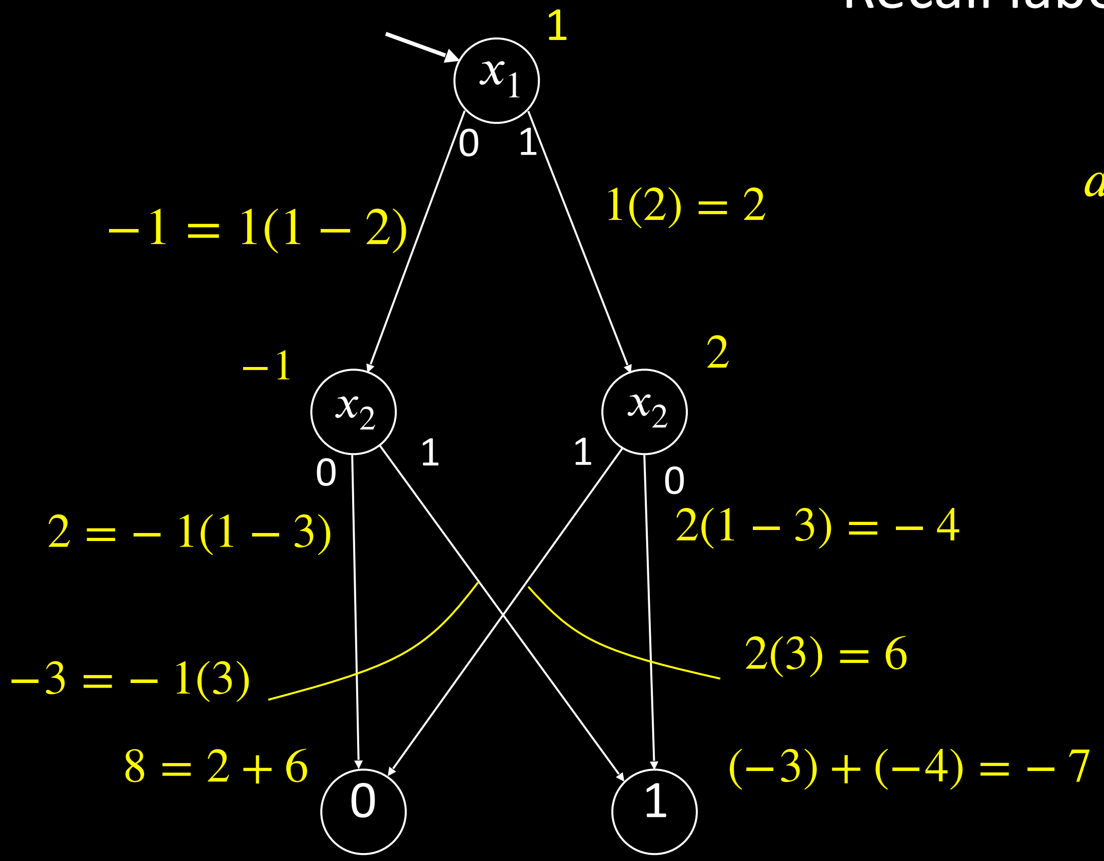
Revised M for EQROBP: "On input $\langle B_1, B_2 \rangle$

1. Pick a random *non-Boolean* input assignment.

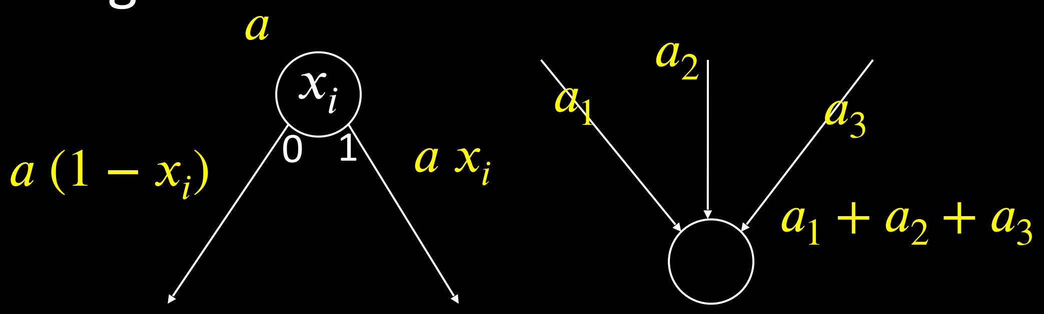
Non-Boolean Inputs

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

Example: $x_1 = 2$, $x_2 = 3$ Output = -7



Recall labeling rules:



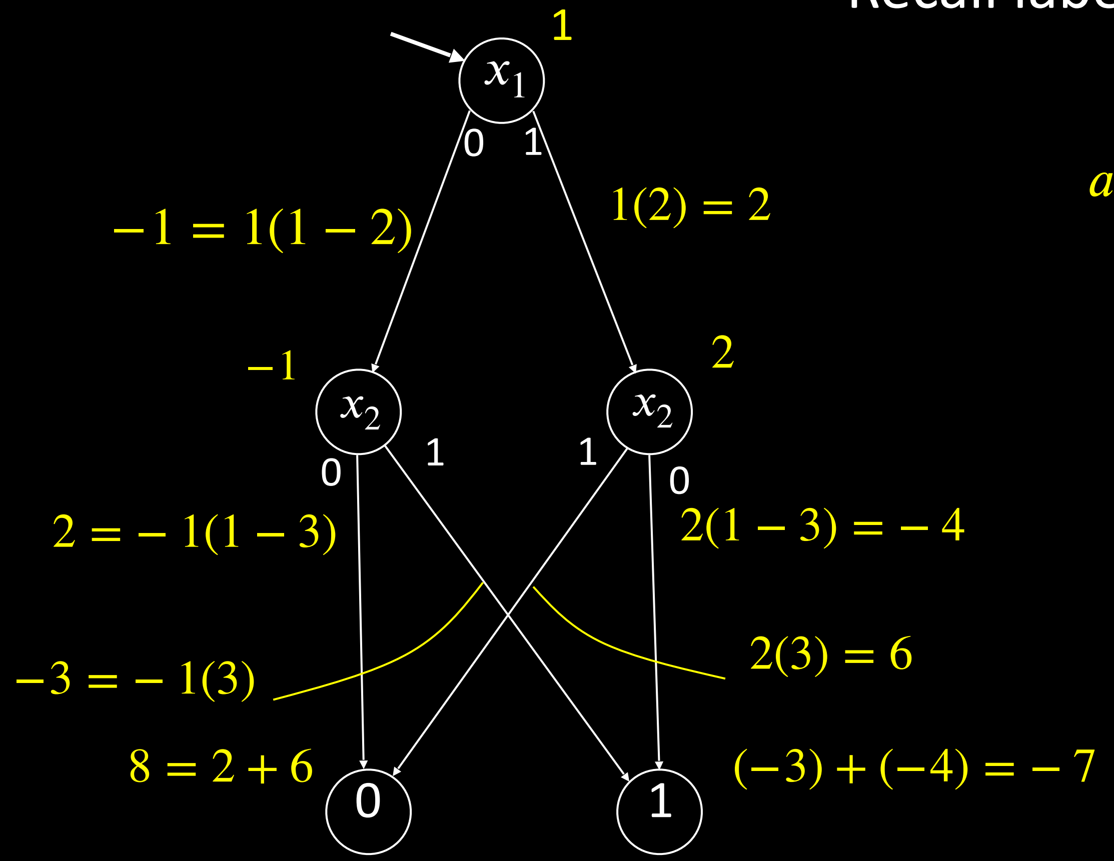
Revised M for EQROBP: "On input $\langle B_1, B_2 \rangle$

1. Pick a random *non-Boolean* input assignment.
2. Evaluate B_1 and B_2 on that assignment.

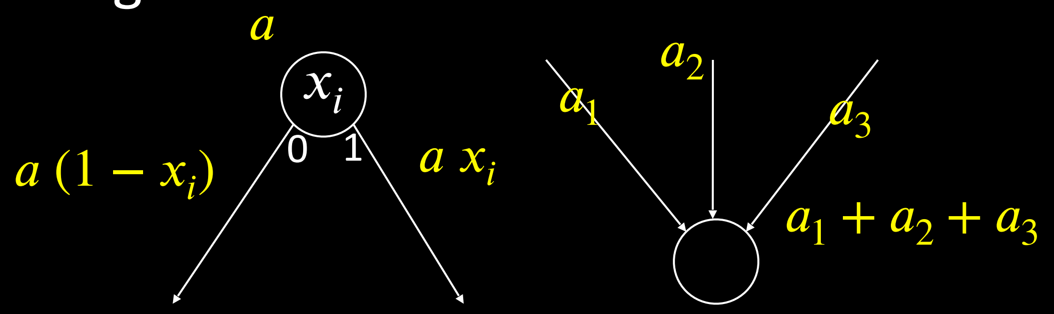
Non-Boolean Inputs

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

Example: $x_1 = 2, \quad x_2 = 3$ Output = -7



Recall labeling rules:



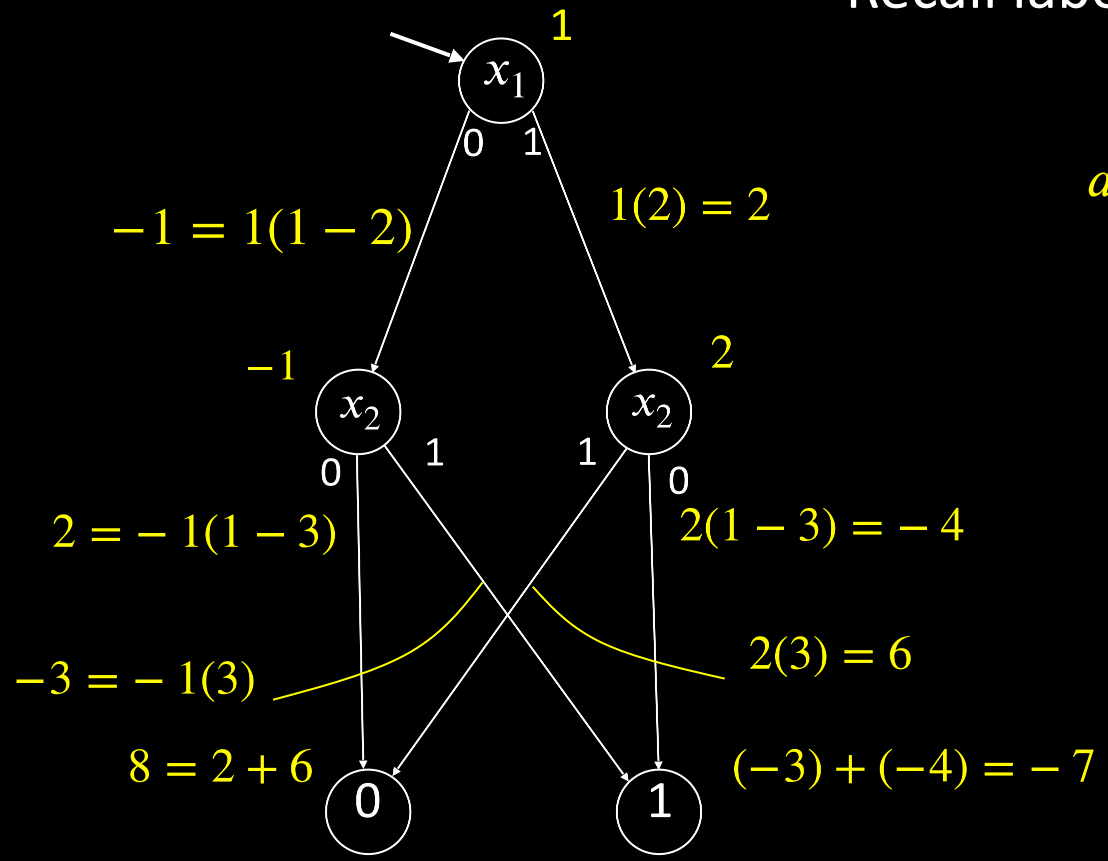
Revised M for EQROBP: "On input $\langle B_1, B_2 \rangle$

1. Pick a random *non-Boolean* input assignment.
2. Evaluate B_1 and B_2 on that assignment.
3. If B_1 and B_2 disagree then *reject*.
   If they agree then *accept*."

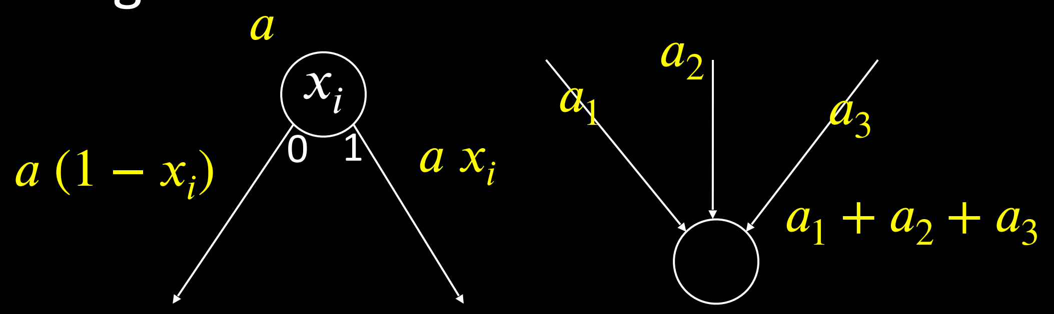
Non-Boolean Inputs

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

Example: $x_1 = 2,\ \ x_2 = 3$ Output = -7



Recall labeling rules:



Revised M for EQROBP: "On input $\langle B_1, B_2\rangle$

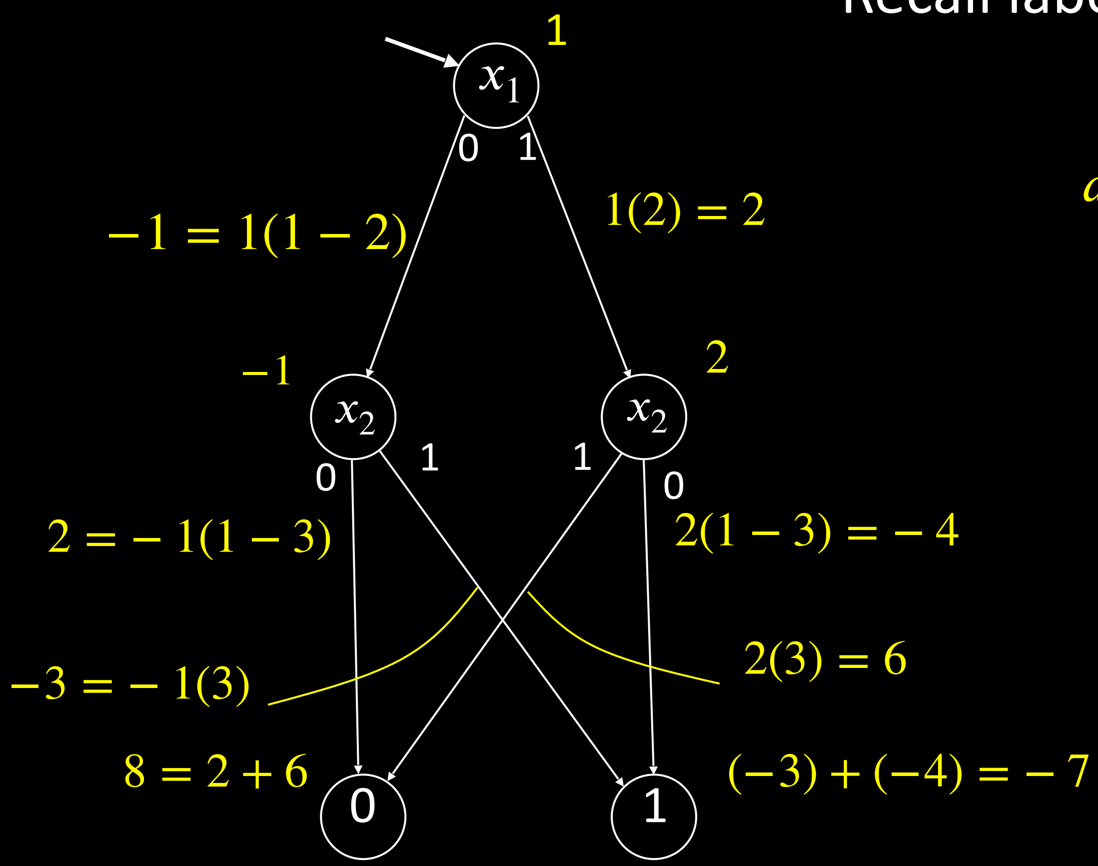
1. Pick a random *non-Boolean* input assignment.
2. Evaluate B_1 and B_2 on that assignment.
3. If B_1 and B_2 disagree then *reject*.
   If they agree then *accept*."

Correctness proof…

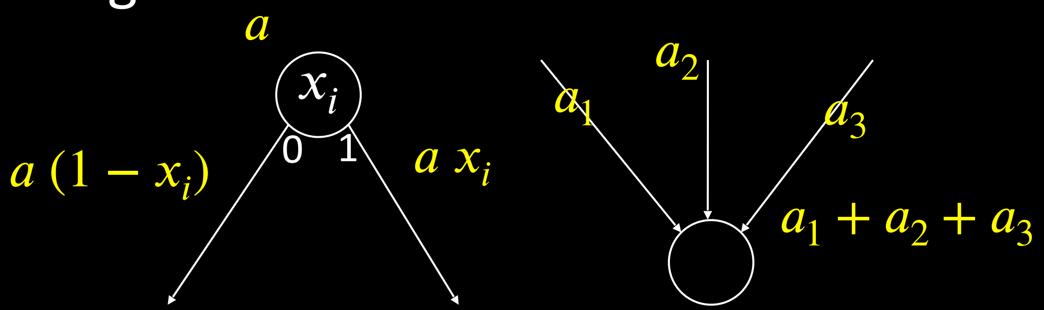
Non-Boolean Inputs

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

Example: $x_1 = 2$, $x_2 = 3$ Output = -7



Recall labeling rules:



Revised M for EQROBP: "On input $\langle B_1, B_2 \rangle$

1. Pick a random *non-Boolean* input assignment.
2. Evaluate B_1 and B_2 on that assignment.
3. If B_1 and B_2 disagree then *reject*.
   If they agree then *accept*."

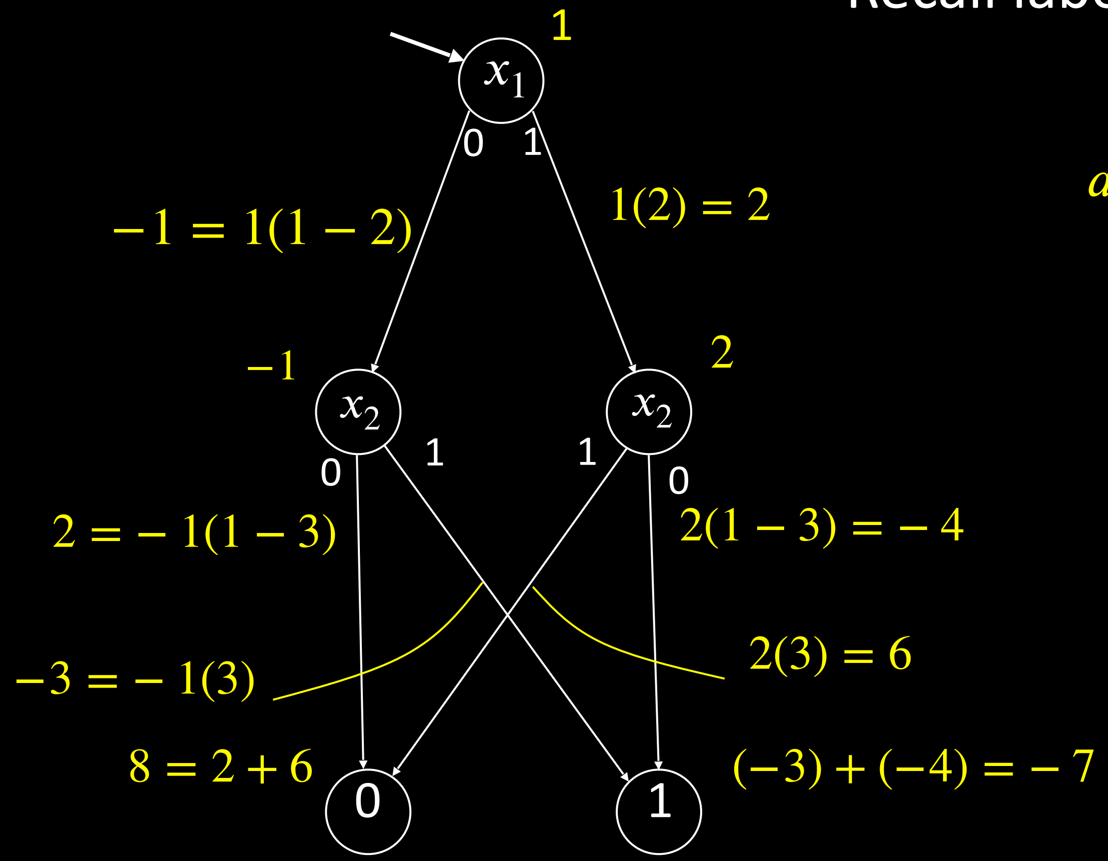
Correctness proof…

Check-in 23.3

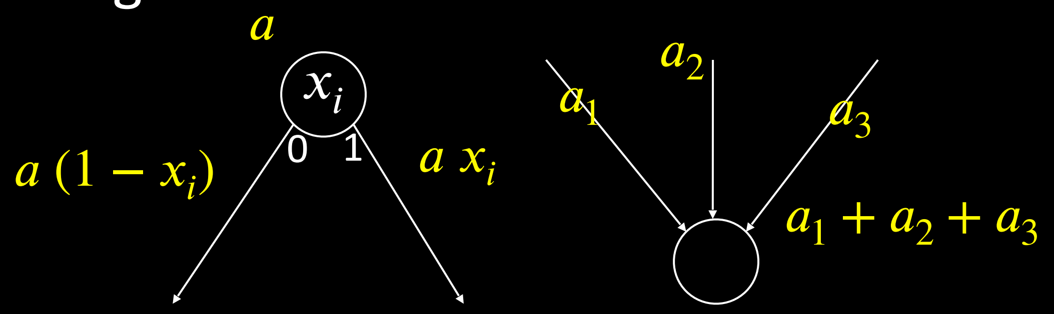
Non-Boolean Inputs

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

Example: $x_1 = 2, \quad x_2 = 3$ Output = -7



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

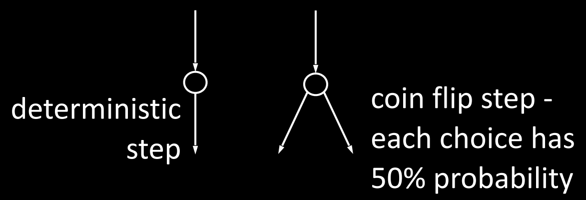
1. Defined probabilistic Turing machines
2. Defined the class BPP
3. Sketched the amplification lemma
4. Introduced branching programs and read-once branching programs
5. Started the proof that EQROBP $\in$ BPP
6. Introduced the arithmetization method

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 EQROBP $\in$ BPP
6. Introduced the arithmetization method

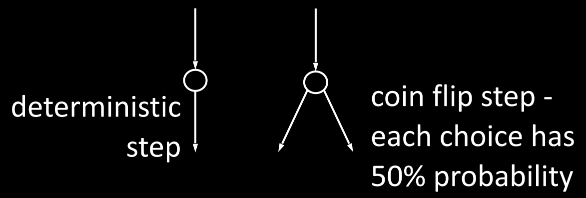
Review: Probabilistic TMs and BPP

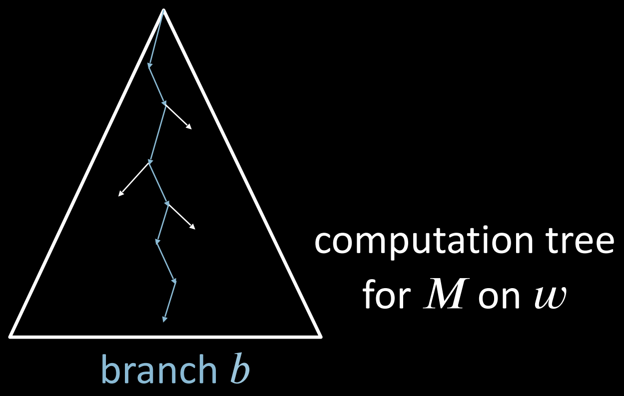
Defn: A <u>probabilistic Turing machine</u> (PTM) is a variant of a NTM where each computation step has 1 or 2 possible choices.



Review: Probabilistic TMs and BPP

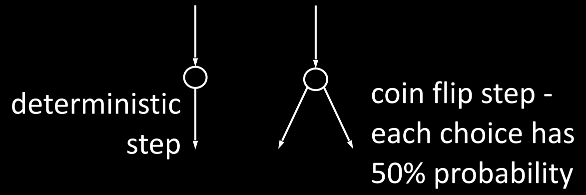
Defn: A <u>probabilistic Turing machine</u> (PTM) is a variant of a NTM where each computation step has 1 or 2 possible choices.

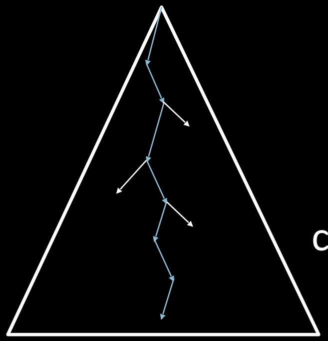




Review: Probabilistic TMs and BPP

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





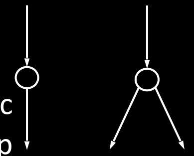
Pr[branch b] $= 2^{-k}$ where b has k coin flips

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

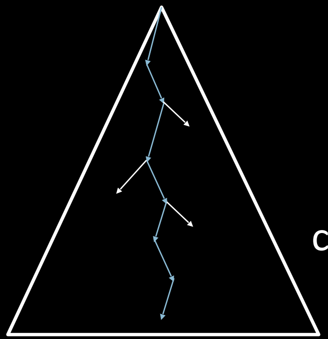
Pr[M rejects w] $= 1 -$ Pr[M accepts w]

Review: Probabilistic TMs and BPP

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



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



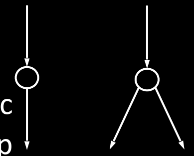
Pr[branch b] $= 2^{-k}$ where b has k coin flips

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

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

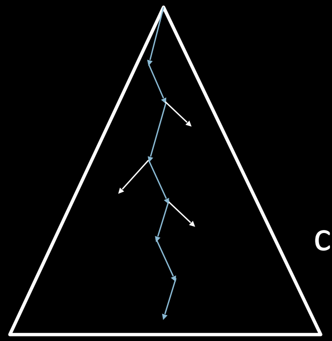
Review: Probabilistic TMs and BPP

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



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

Defn: BPP $= \{ A \mid$ some poly-time PTM decides A with error $\epsilon = \frac{1}{3} \}$



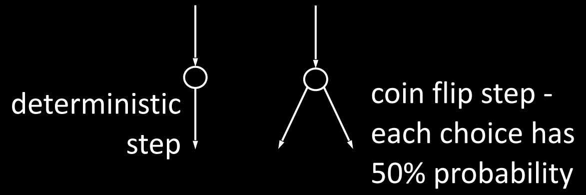
Pr[branch b] $= 2^{-k}$ where b has k coin flips

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

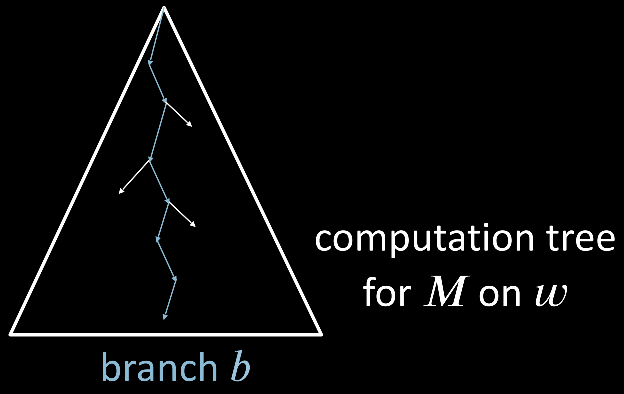
Pr[M rejects w] $= 1 -$ Pr[M accepts w]

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



Defn: BPP $= \{ A \mid$ some poly-time PTM decides A with error $\epsilon = \frac{1}{3} \}$

$\Downarrow$

Amplification lemma: $2^{-\text{poly}(n)}$

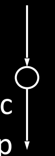
Pr[branch b] $= 2^{-k}$ where b has k coin flips

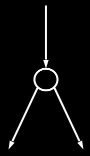
Pr[M accepts w] $= \sum_{b \text{ accepts}}$ Pr[branch b]

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

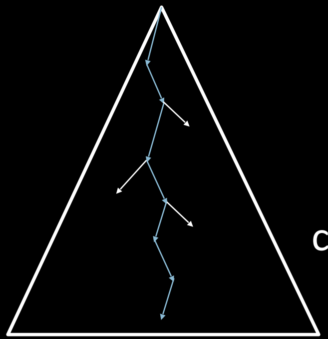
Review: Probabilistic TMs and BPP

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





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



Defn: BPP $= \{A \mid$ some poly-time PTM decides A with error $\epsilon = \frac{1}{3}$ $\}$

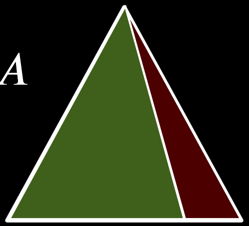
$\Downarrow$

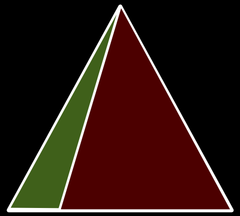
Amplification lemma: $2^{-\text{poly}(n)}$

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

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

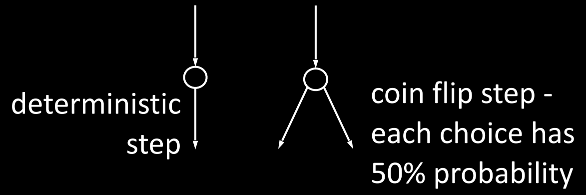
Pr[M rejects w] $= 1 -$ Pr[M accepts w]



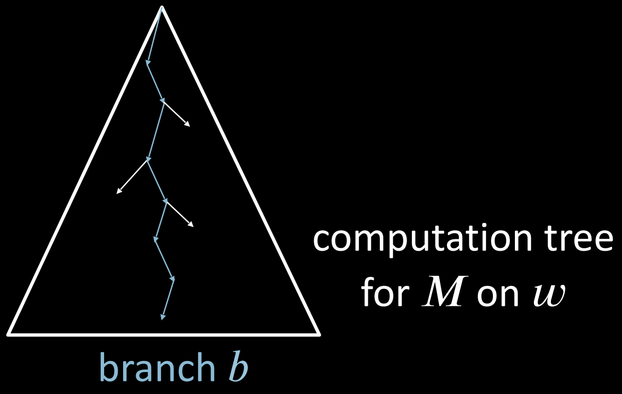


Review: Probabilistic TMs and BPP

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



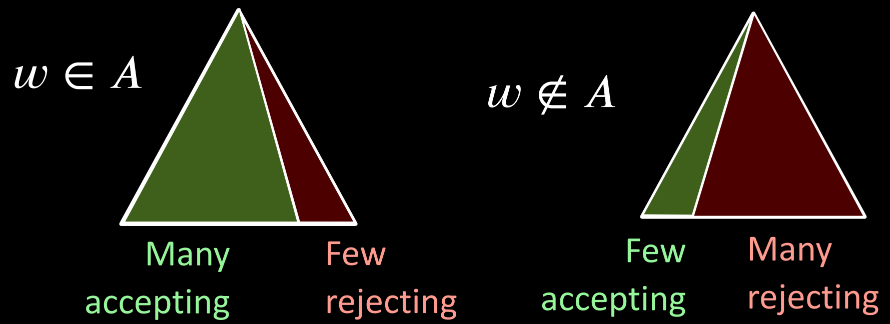
Defn: For $\epsilon \geq 0$ say PTM M <u>decides language A with error probability ϵ</u> if for every w, Pr[M gives the wrong answer about $w \in A$] $\leq \epsilon$.



Defn: BPP $= \{ A \mid$ some poly-time PTM decides A with error $\epsilon = \frac{1}{3} \}$

$\Downarrow$

Amplification lemma: $2^{-\text{poly}(n)}$



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

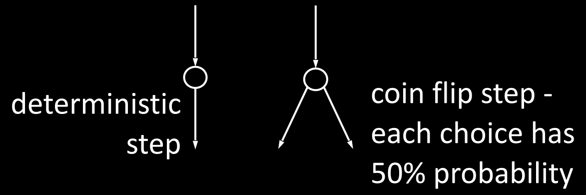
Pr[M accepts w] $= \sum_{b \text{ accepts}}$ Pr[branch b]

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

Check-in 24.1

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.

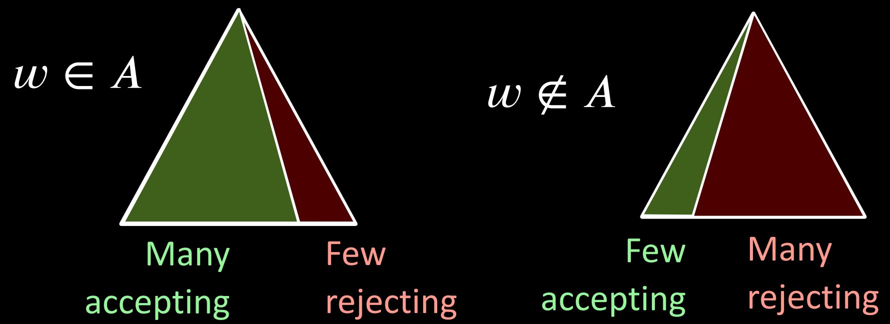


Defn: For $\epsilon \geq 0$ say PTM M decides language A with error probability ϵ if for every w, Pr[M gives the wrong answer about $w \in A$] $\leq \epsilon$.

Defn: BPP $= \{A \mid$ some poly-time PTM decides A with error $\epsilon = \frac{1}{3}$

⇩

Amplification lemma: 2^{-p}



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 branching program (BP) is a directed, acyclic (no cycles) graph that has

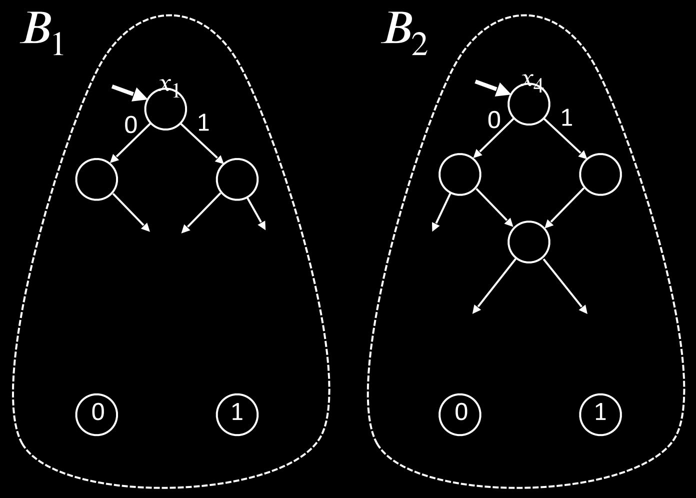
1. *Query nodes* labeled x_i and having two outgoing edges labeled 0 and 1.
2. *Two output nodes* labeled 0 and 1 and having no outgoing edges.
3. A designated *start node*.

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)



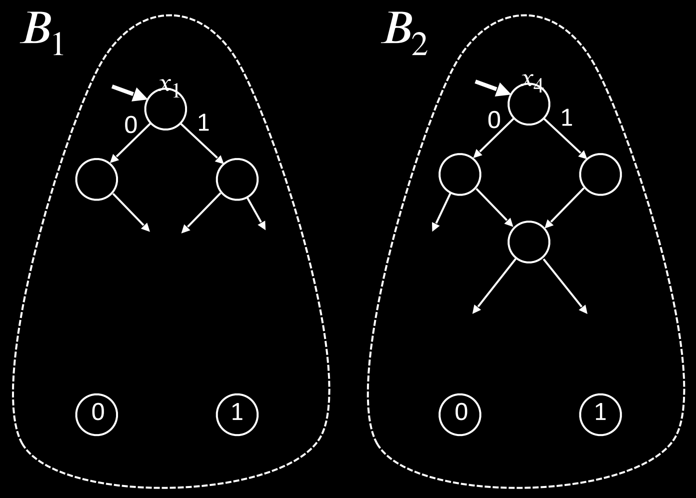
Review: Branching Programs

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.



Review: Branching Programs

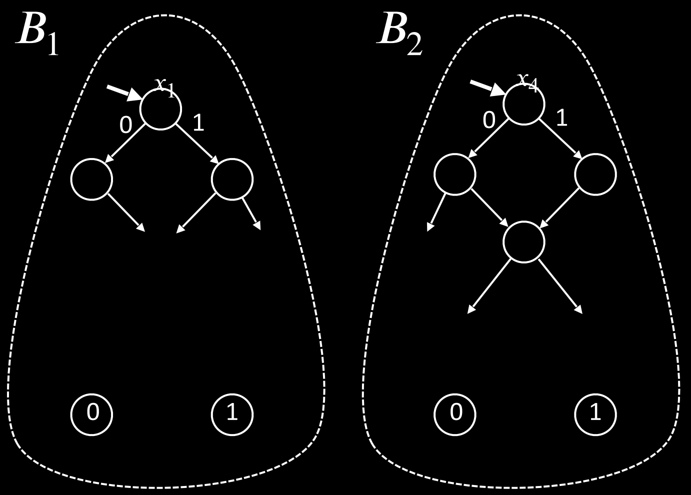
Defn: A <u>branching program</u> (BP) is a directed, acyclic (no cycles) graph that has

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**ROBP** $= \{\langle B_1, B_2\rangle \mid$ B_1 and B_2 are equivalent read-once BPs$\}$



Review: Branching Programs

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

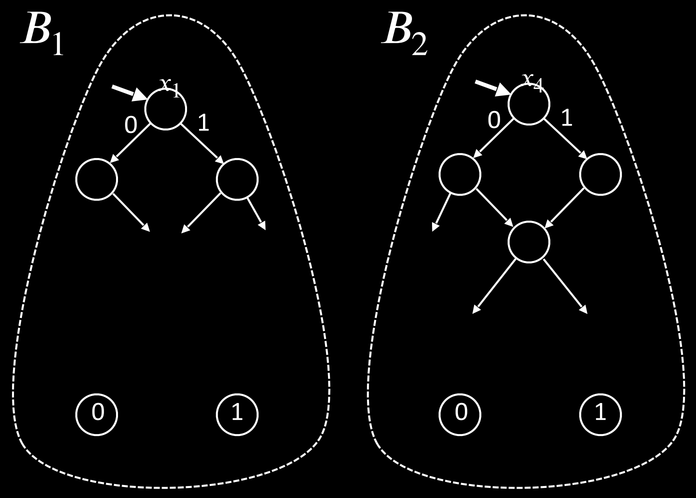
1. *Query nodes* labeled x_i and having two outgoing edges labeled 0 and 1.
2. *Two output nodes* labeled 0 and 1 and having no outgoing edges.
3. A designated *start node*.

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**ROBP** $= \{\langle B_1, B_2\rangle \mid B_1$ and B_2 are equivalent read-once BPs$\}$

Theorem: EQ**ROBP** $\in$ BPP



Review: Branching Programs

Defn: A branching program (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*.

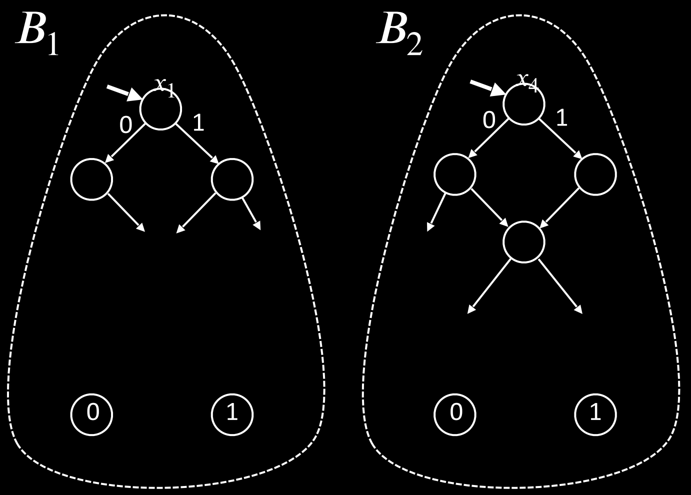
Theorem: EQ**BP** is coNP-complete (on pset 6)

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

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

Theorem: EQ**ROBP** $\in$ BPP

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



Review: Branching Programs

Defn: A branching program (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)

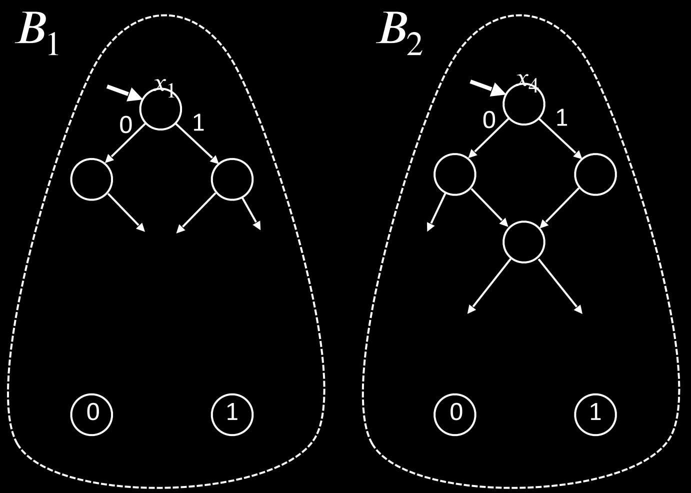
Defn: A BP is read-once if it never queries a variable more than once on any path from the start node to an output.

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

Theorem: EQ**ROBP** $\in$ BPP

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

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



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)

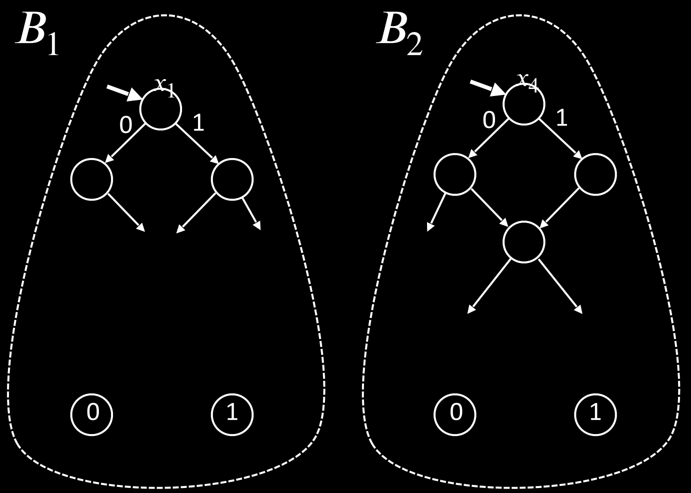
Defn: A BP is <u>read-once</u> if it never queries a variable more than once on any path from the start node to an output.

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

Theorem: EQ**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.

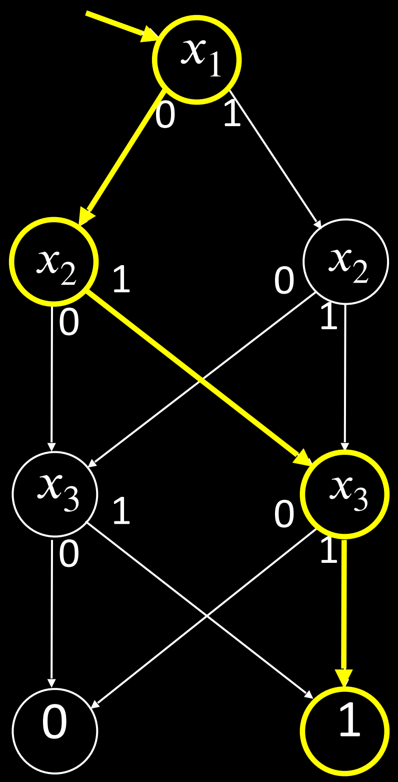


Boolean Labeling

Alternative way to view BP computation

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

The BP follows its execution path.



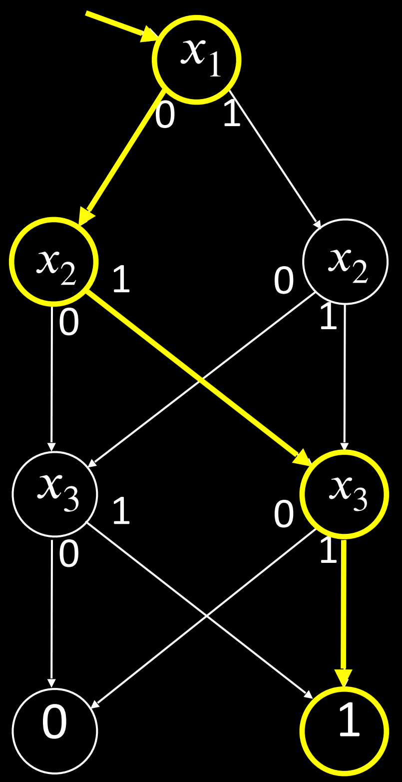
Boolean Labeling

Alternative way to view BP computation

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

The BP follows its execution path.

Label all nodes and edges on the execution path with 1



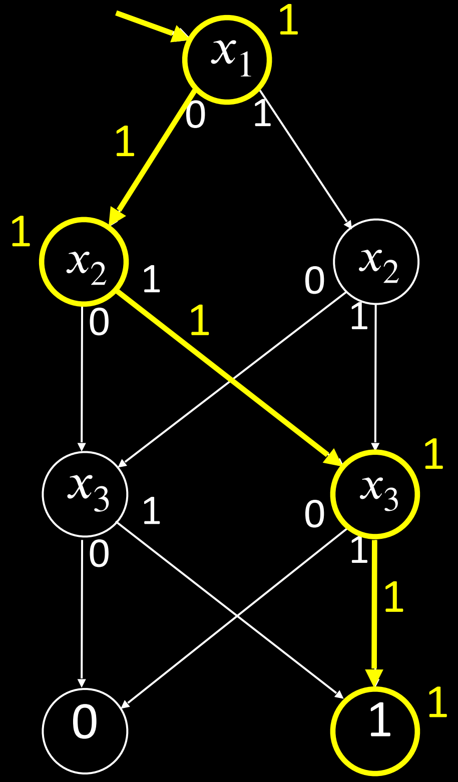
Boolean Labeling

Alternative way to view BP computation

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

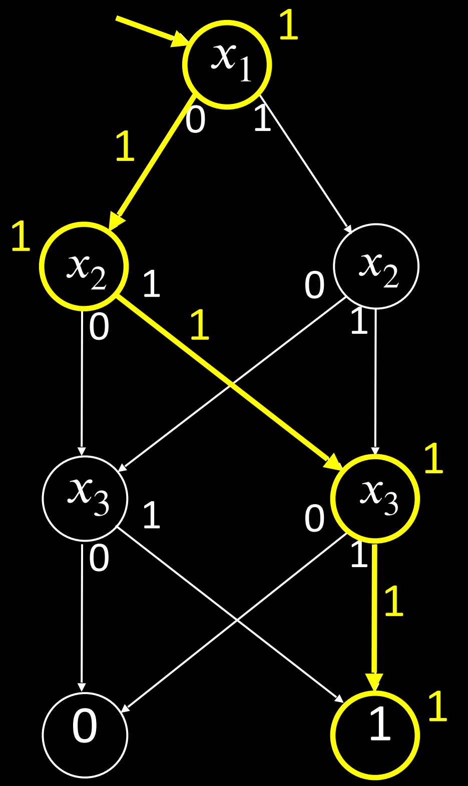
The BP follows its execution path.

Label all nodes and edges on the execution path with 1



Boolean Labeling

Alternative way to view BP computation



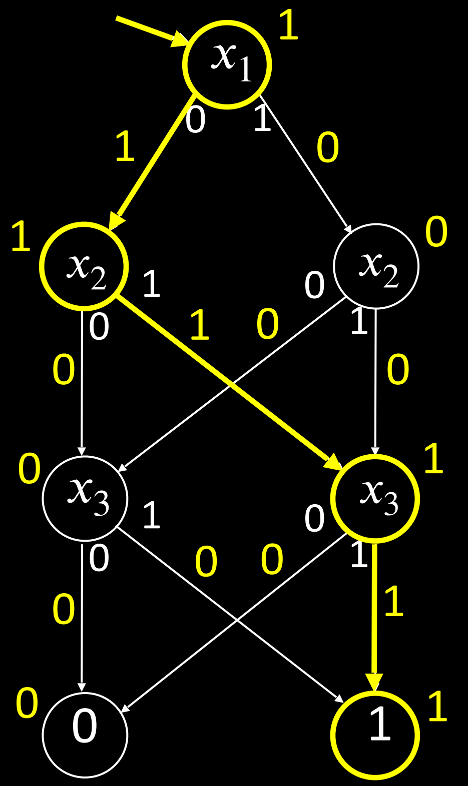
Show by example: Input is $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$

The BP follows its execution path.

Label all nodes and edges on the execution path with 1 and off the execution path with 0.

Boolean Labeling

Alternative way to view BP computation



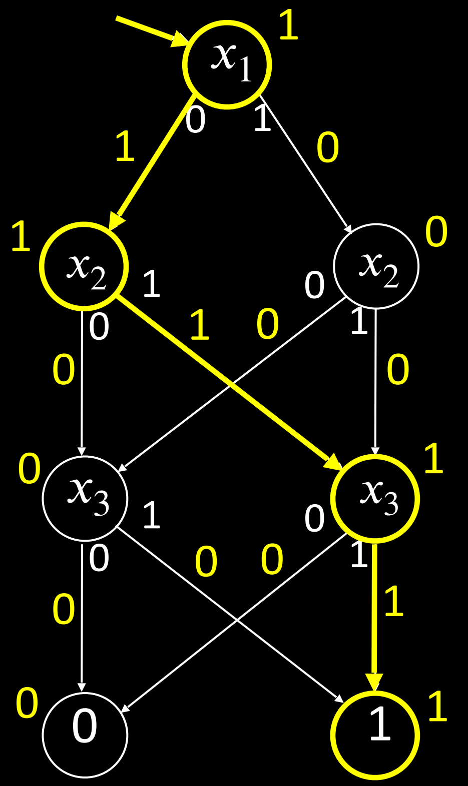
Show by example: Input is $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$

The BP follows its execution path.

Label all nodes and edges on the execution path with 1 and off the execution path with 0.

Boolean Labeling

Alternative way to view BP computation



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

The BP follows its execution path.

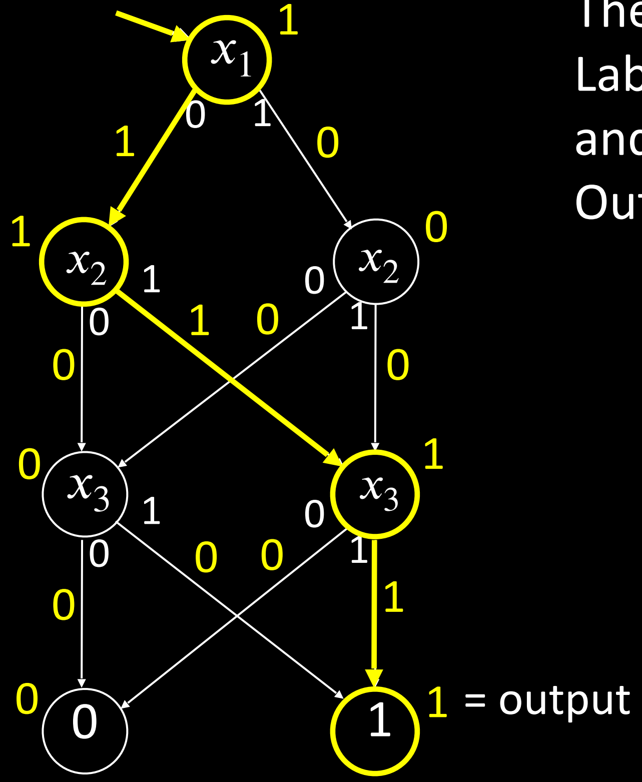
Label all nodes and edges on the execution path with 1

and off the execution path with 0.

Output the label of the output node 1.

Boolean Labeling

Alternative way to view BP computation



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

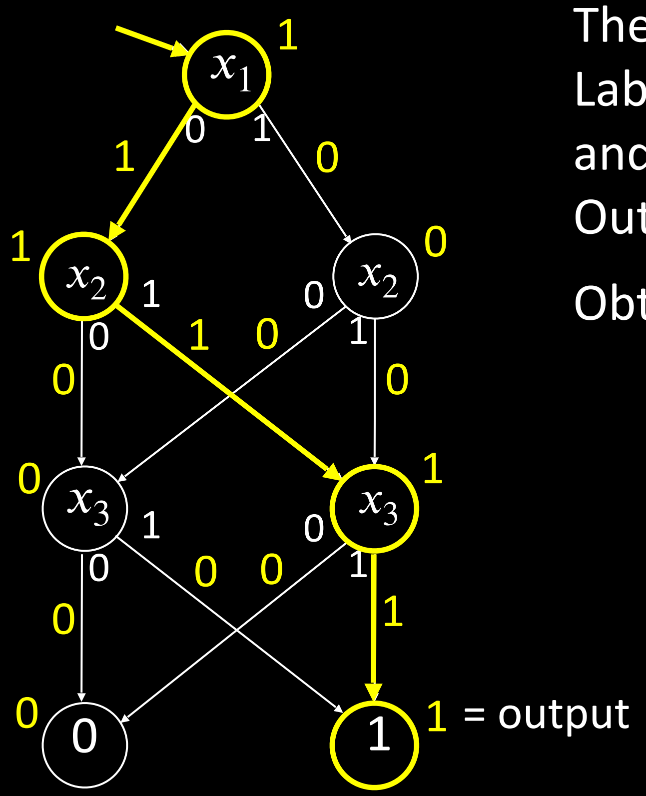
The BP follows its execution path.

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.

Boolean Labeling

Alternative way to view BP computation



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

The BP follows its execution path.

Label all nodes and edges on the execution path with 1

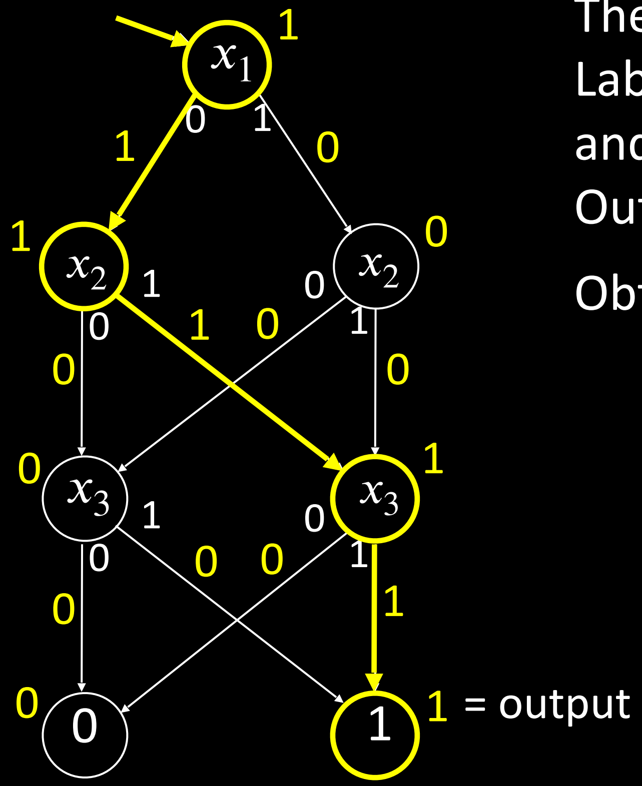
and off the execution path with 0.

Output the label of the output node 1.

Obtain the labeling inductively by using these rules:

Boolean Labeling

Alternative way to view BP computation



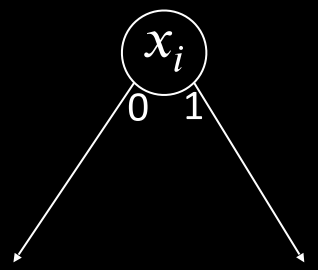
Show by example: Input is $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$

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.

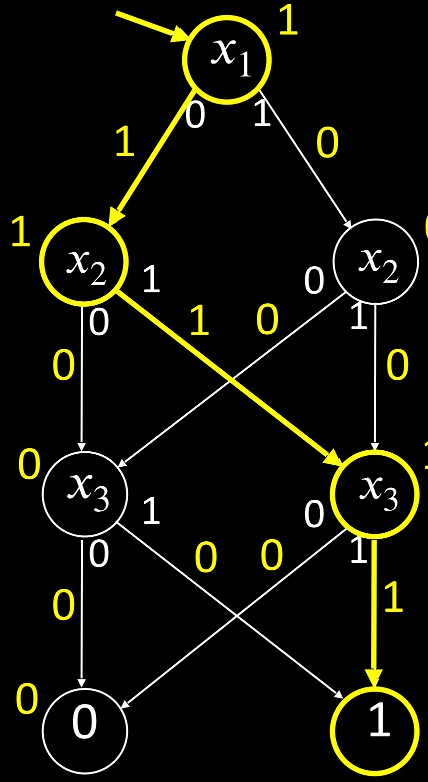
Obtain the labeling inductively by using these rules:



Label outgoing edges from nodes

Boolean Labeling

Alternative way to view BP computation



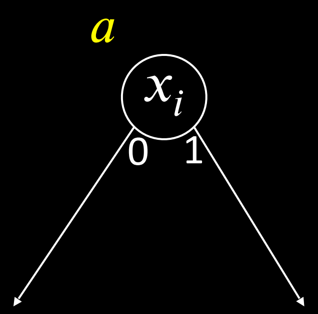
Show by example: Input is $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$

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

Boolean Labeling

Alternative way to view BP computation

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

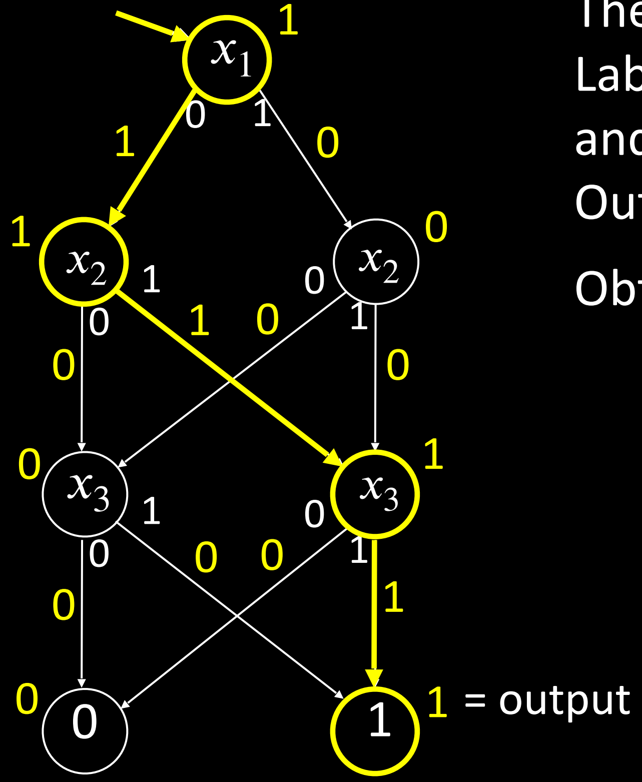
The BP follows its execution path.

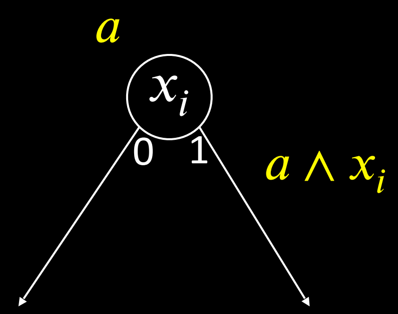
Label all nodes and edges on the execution path with 1

and off the execution path with 0.

Output the label of the output node 1.

Obtain the labeling inductively by using these rules:

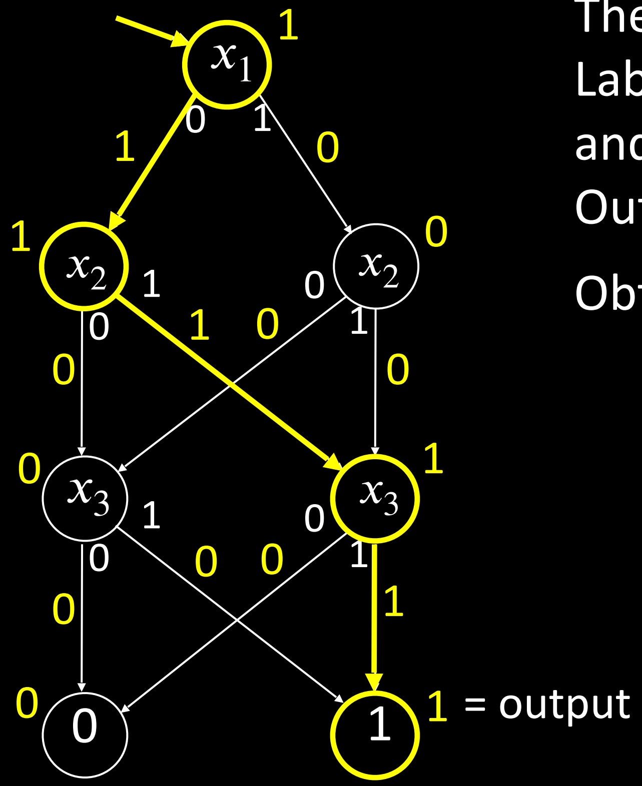




Label outgoing edges from nodes

Boolean Labeling

Alternative way to view BP computation



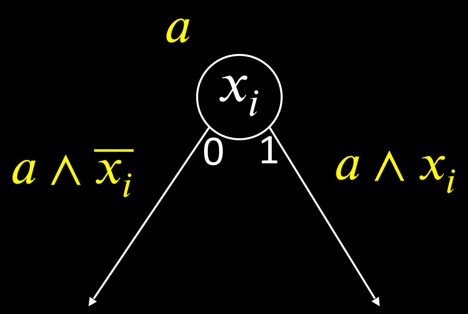
Show by example: Input is $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$

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.

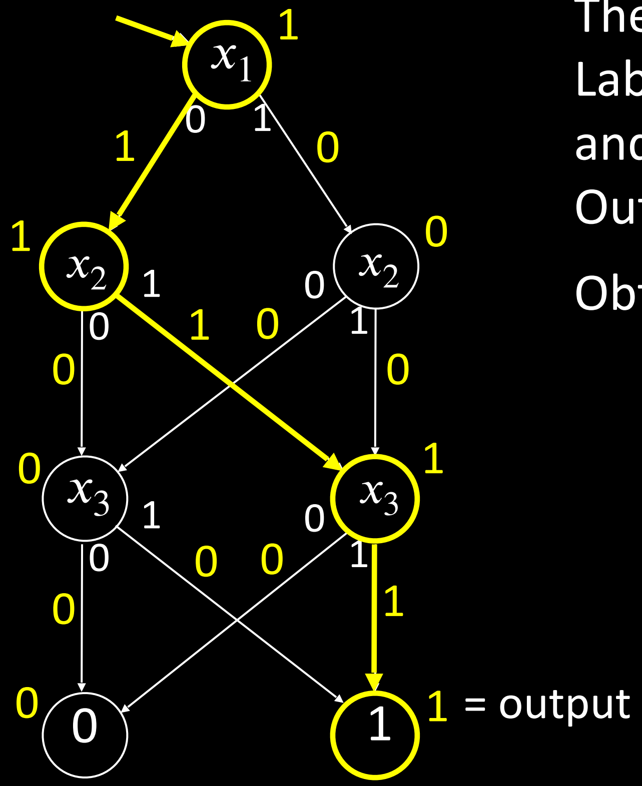
Obtain the labeling inductively by using these rules:



Label outgoing edges from nodes

Boolean Labeling

Alternative way to view BP computation



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

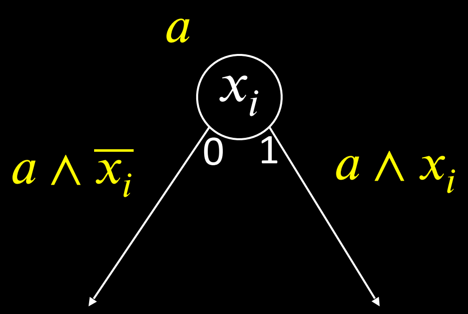
The BP follows its execution path.

Label all nodes and edges on the execution path with 1

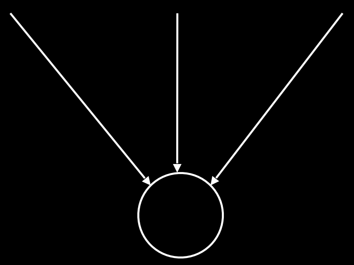
and off the execution path with 0.

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

Boolean Labeling

Alternative way to view BP computation

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

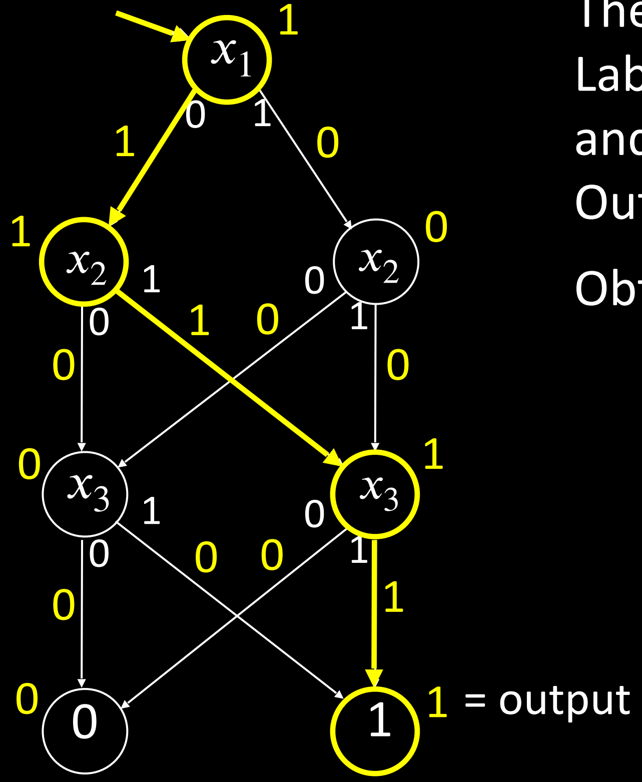
The BP follows its execution path.

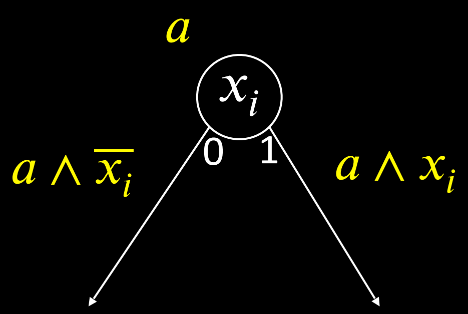
Label all nodes and edges on the execution path with 1

and off the execution path with 0.

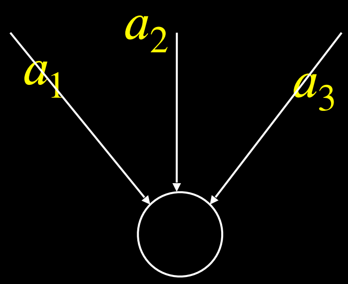
Output the label of the output node 1.

Obtain the labeling inductively by using these rules:





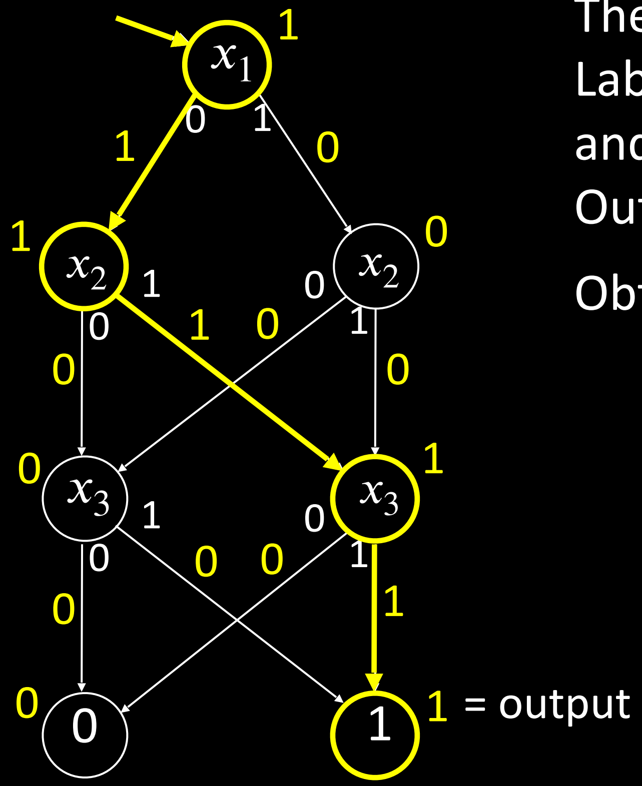
Label outgoing edges from nodes



Label nodes from incoming edges

Boolean Labeling

Alternative way to view BP computation



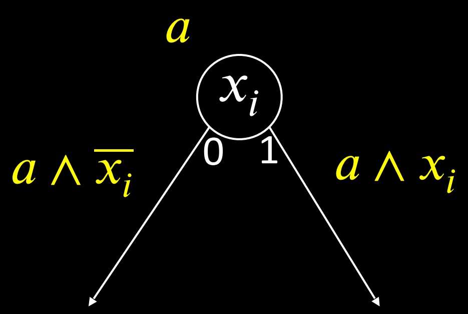
Show by example: Input is $x_1 = 0, \quad x_2 = 1, \quad x_3 = 1$

The BP follows its execution path.

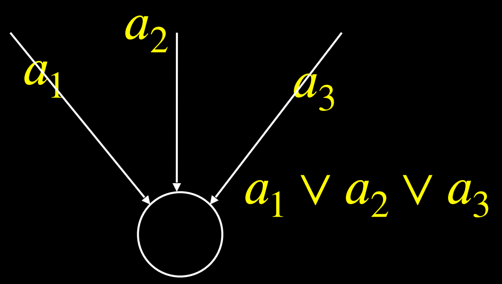
Label all nodes and edges on the execution path with 1 and off the execution path with 0.

Output the label of the output node 1.

Obtain the labeling inductively by using these rules:



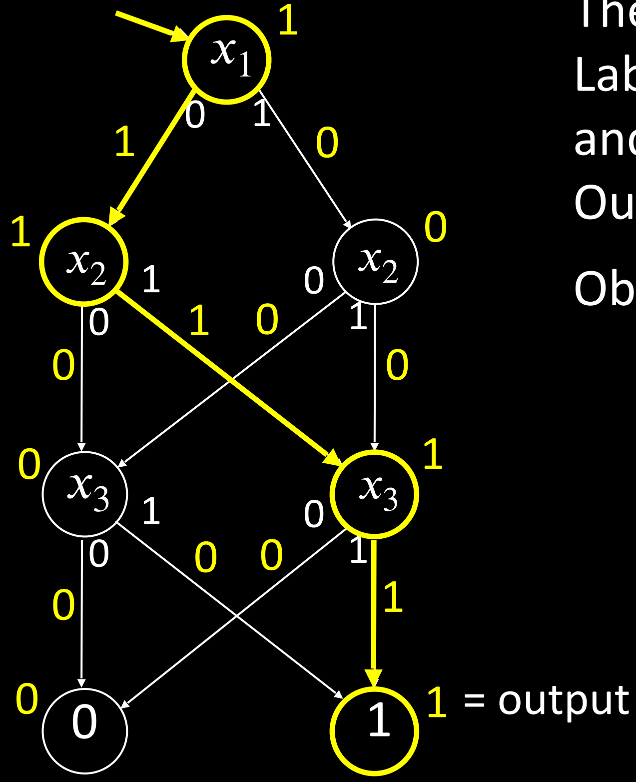
Label outgoing edges from nodes



Label nodes from incoming edges

Boolean Labeling

Alternative way to view BP computation



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

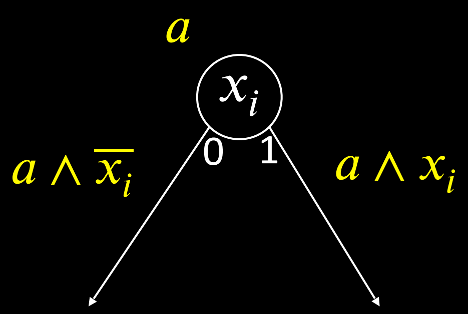
The BP follows its execution path.

Label all nodes and edges on the execution path with 1

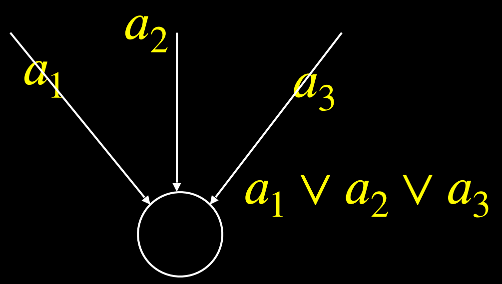
and off the execution path with 0.

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 $\wedge$ and $\vee$ with $+$ and $\times$.

Arithmetization Method

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

$$a \wedge b \quad \rightarrow \quad a \times b = ab$$

Arithmetization Method

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

$$a \wedge b \quad \rightarrow \quad a \times b = ab$$
$$\overline{a} \quad \rightarrow \quad (1 - a)$$

Arithmetization Method

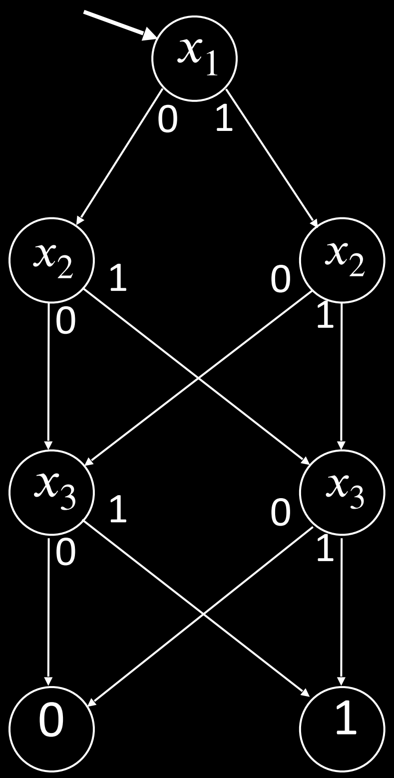
Method: Simulate $\wedge$ and $\vee$ with $+$ and $\times$.

$$\begin{aligned} a \wedge b \quad &\rightarrow \quad a \times b = ab \\ \overline{a} \quad &\rightarrow \quad (1 - a) \\ a \vee b \quad &\rightarrow \quad a + b - ab \end{aligned}$$

Arithmetization Method

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

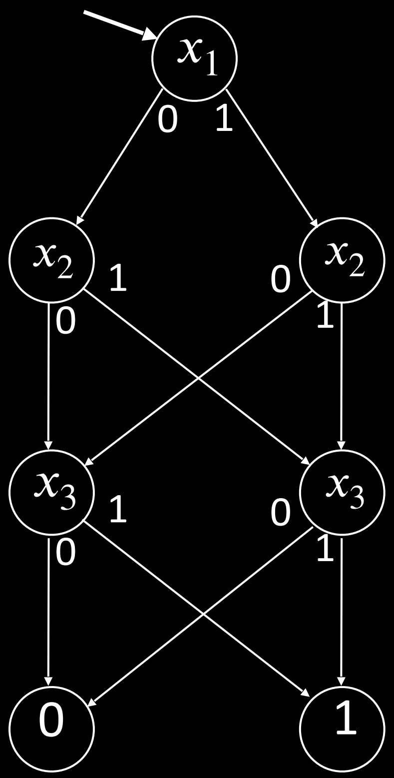
$$a \wedge b \quad \rightarrow \quad a \times b = ab$$
$$\overline{a} \quad \rightarrow \quad (1 - a)$$
$$a \vee b \quad \rightarrow \quad a + b - ab$$



Arithmetization Method

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

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

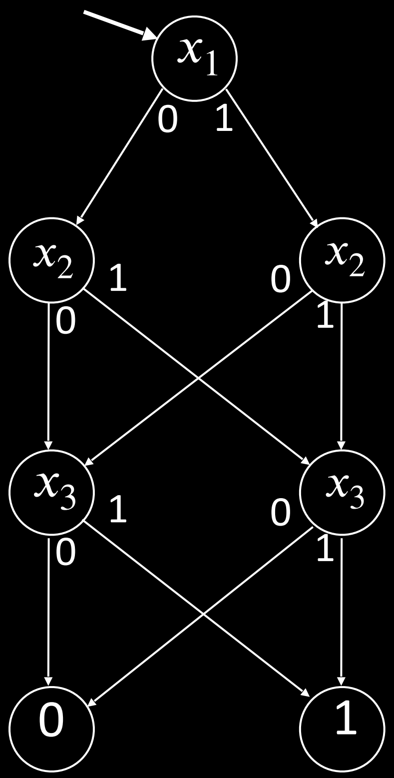


Replace Boolean labeling with arithmetical labeling

Arithmetization Method

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

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



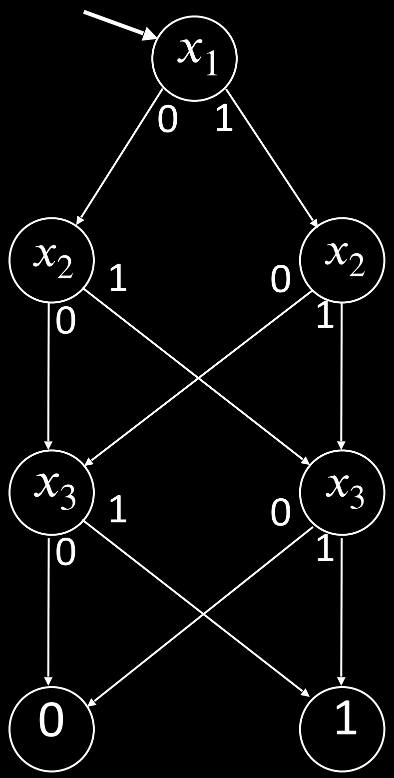
Replace Boolean labeling with arithmetical labeling

Inductive rules:

Arithmetization Method

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

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



Replace Boolean labeling with arithmetical labeling

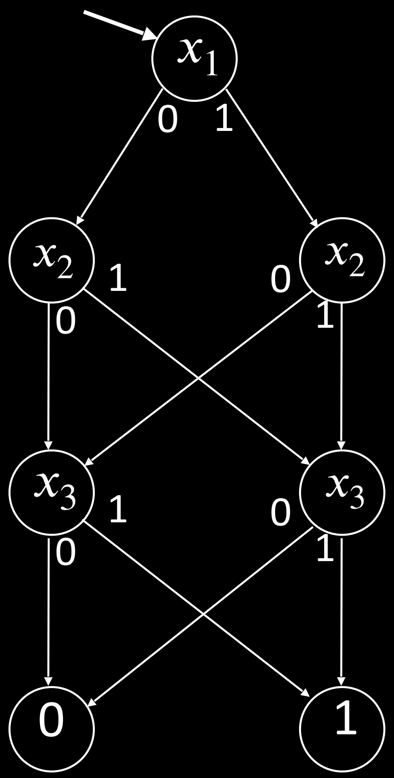
Inductive rules:

Start node labeled 1

Arithmetization Method

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

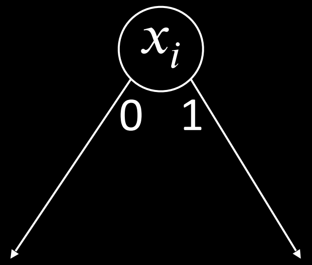
$$a \wedge b \;\rightarrow\; a \times b = ab$$
$$\overline{a} \;\rightarrow\; (1-a)$$
$$a \vee b \;\rightarrow\; a + b - ab$$



Replace Boolean labeling with arithmetical labeling

Inductive rules:

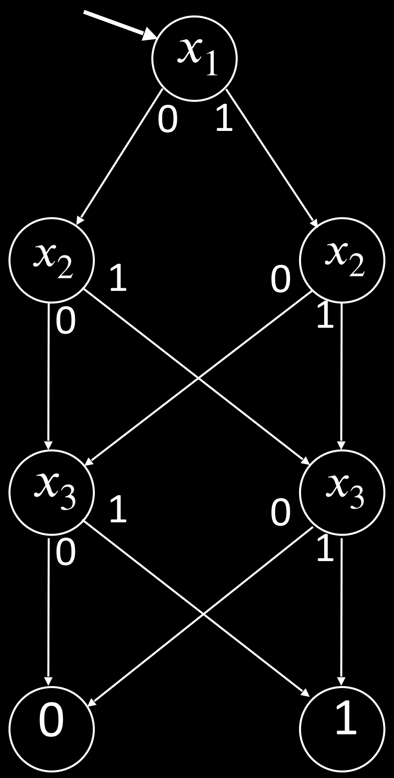
Start node labeled 1



Arithmetization Method

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

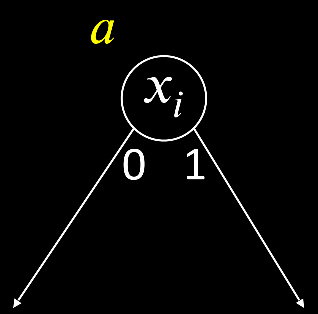
$$a \wedge b \quad \rightarrow \quad a \times b = ab$$
$$\overline{a} \quad \rightarrow \quad (1 - a)$$
$$a \vee b \quad \rightarrow \quad a + b - ab$$



Replace Boolean labeling with arithmetical labeling

Inductive rules:

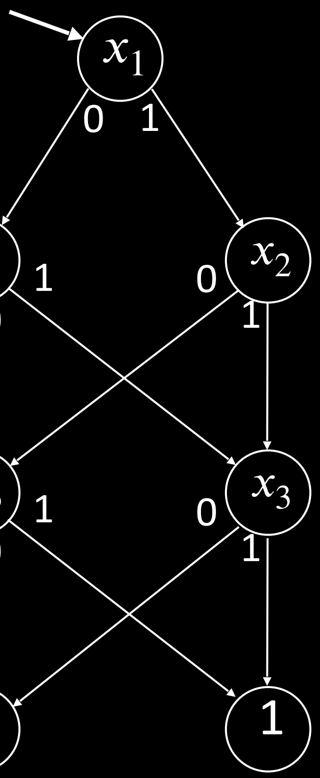
Start node labeled 1



Arithmetization Method

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

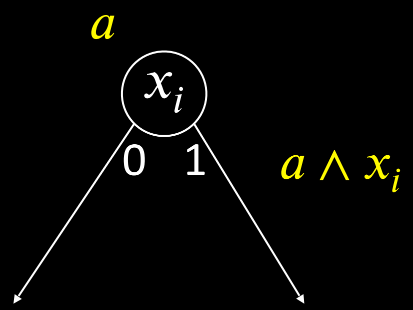
$$a \wedge b \quad \rightarrow \quad a \times b = ab$$
$$\overline{a} \quad \rightarrow \quad (1 - a)$$
$$a \vee b \quad \rightarrow \quad a + b - ab$$



Replace Boolean labeling with arithmetical labeling

Inductive rules:

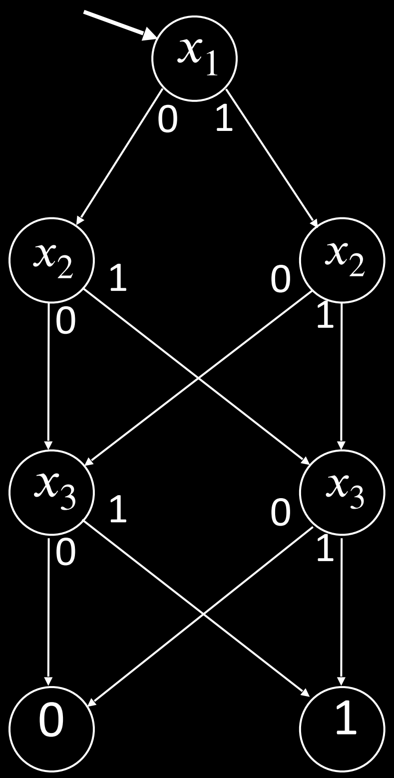
Start node labeled 1



Arithmetization Method

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

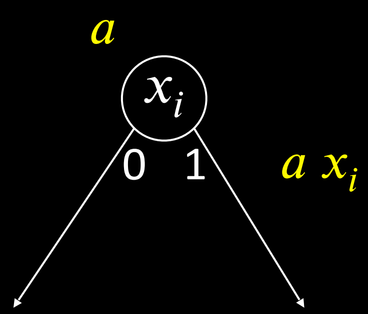
$$a \wedge b \quad \rightarrow \quad a \times b = ab$$
$$\overline{a} \quad \rightarrow \quad (1 - a)$$
$$a \vee b \quad \rightarrow \quad a + b - ab$$



Replace Boolean labeling with arithmetical labeling

Inductive rules:

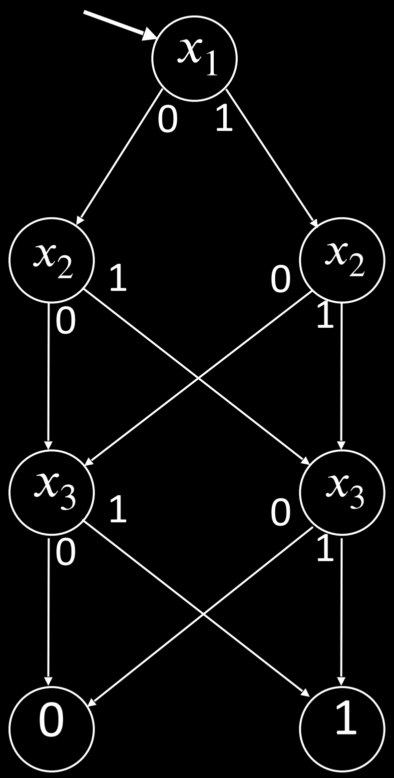
Start node labeled 1



Arithmetization Method

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

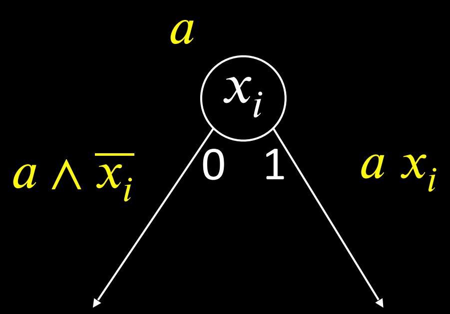
$$a \wedge b \quad \rightarrow \quad a \times b = ab$$
$$\overline{a} \quad \rightarrow \quad (1 - a)$$
$$a \vee b \quad \rightarrow \quad a + b - ab$$



Replace Boolean labeling with arithmetical labeling

Inductive rules:

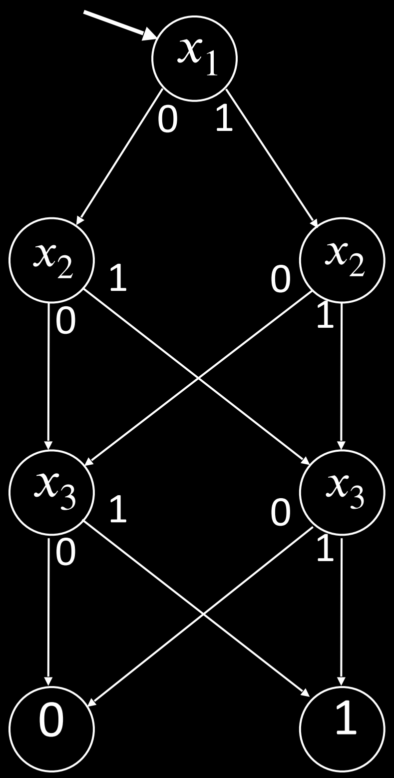
Start node labeled 1



Arithmetization Method

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

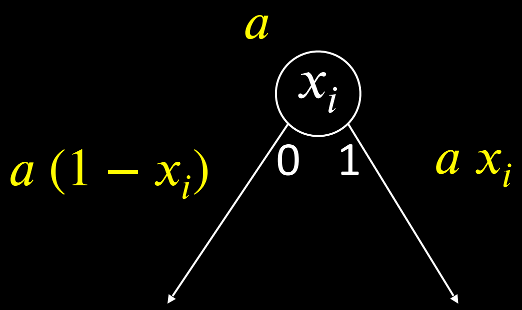
$$a \wedge b \quad \rightarrow \quad a \times b = ab$$
$$\overline{a} \quad \rightarrow \quad (1 - a)$$
$$a \vee b \quad \rightarrow \quad a + b - ab$$



Replace Boolean labeling with arithmetical labeling

Inductive rules:

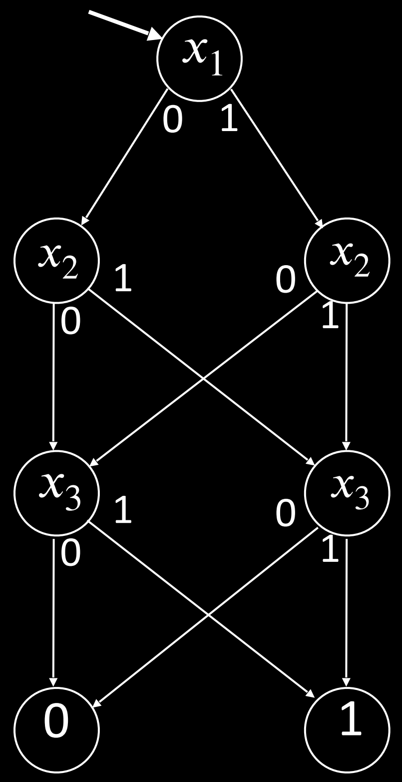
Start node labeled 1



Arithmetization Method

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

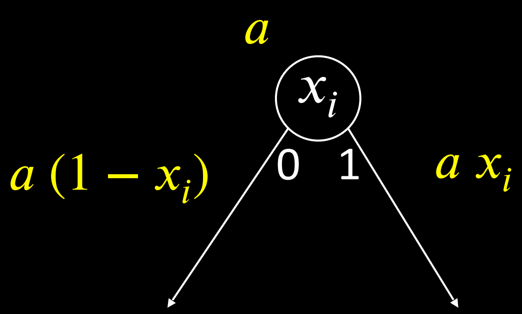
$$a \wedge b \quad \rightarrow \quad a \times b = ab$$
$$\overline{a} \quad \rightarrow \quad (1 - a)$$
$$a \vee b \quad \rightarrow \quad a + b - ab$$

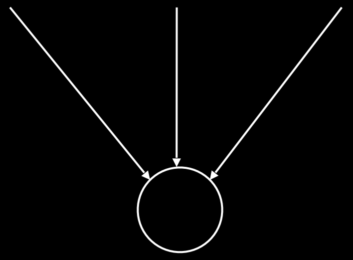


Replace Boolean labeling with arithmetical labeling

Inductive rules:

Start node labeled 1

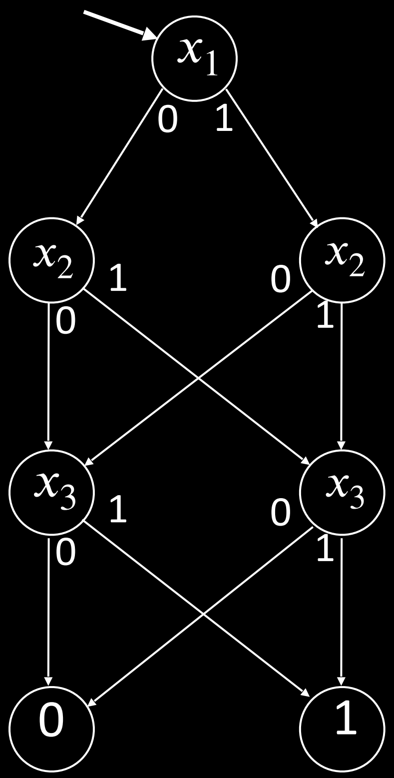




Arithmetization Method

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

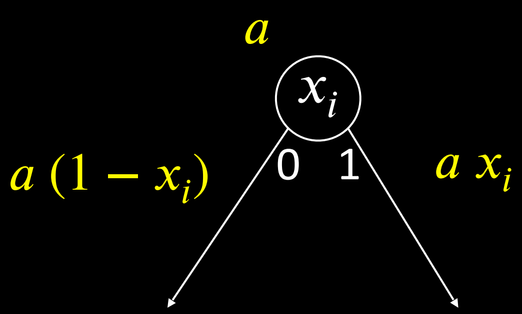
$$a \wedge b \quad \to \quad a \times b = ab$$
$$\overline{a} \quad \to \quad (1 - a)$$
$$a \vee b \quad \to \quad a + b - ab$$

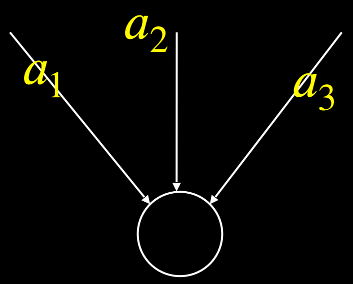


Replace Boolean labeling with arithmetical labeling

Inductive rules:

Start node labeled 1

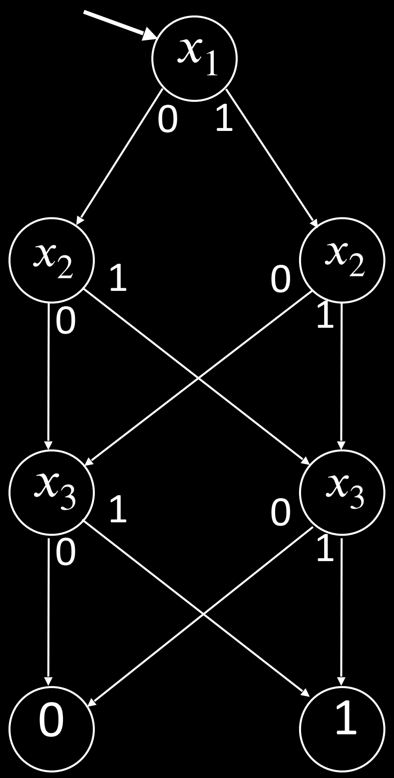




Arithmetization Method

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

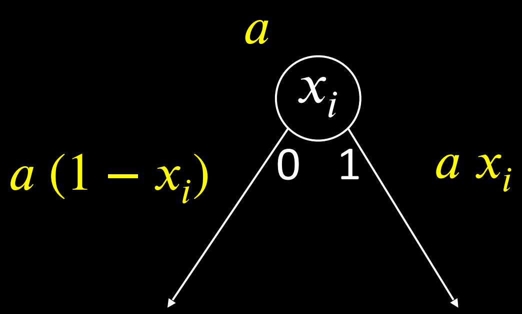
$$a \wedge b \;\rightarrow\; a \times b = ab$$
$$\overline{a} \;\rightarrow\; (1 - a)$$
$$a \vee b \;\rightarrow\; a + b - ab$$

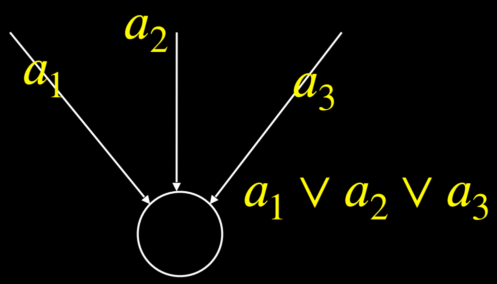


Replace Boolean labeling with arithmetical labeling

Inductive rules:

Start node labeled 1

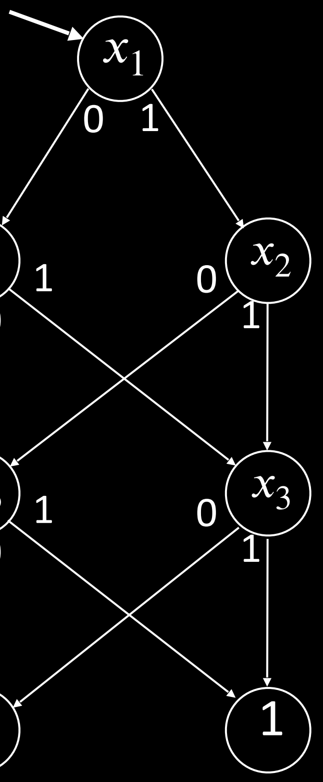




Arithmetization Method

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

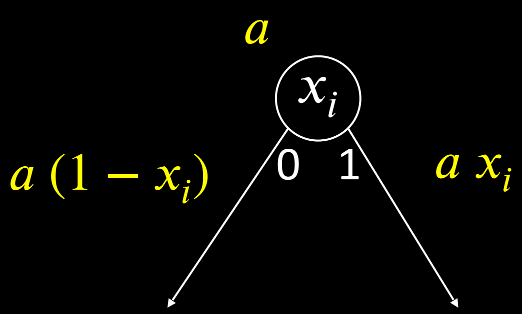
$$a \wedge b \quad \to \quad a \times b = ab$$
$$\overline{a} \quad \to \quad (1 - a)$$
$$a \vee b \quad \to \quad a + b - ab$$

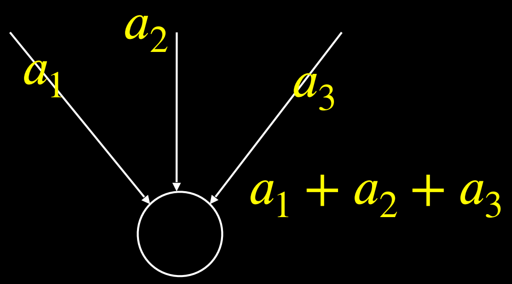


Replace Boolean labeling with arithmetical labeling

Inductive rules:

Start node labeled 1

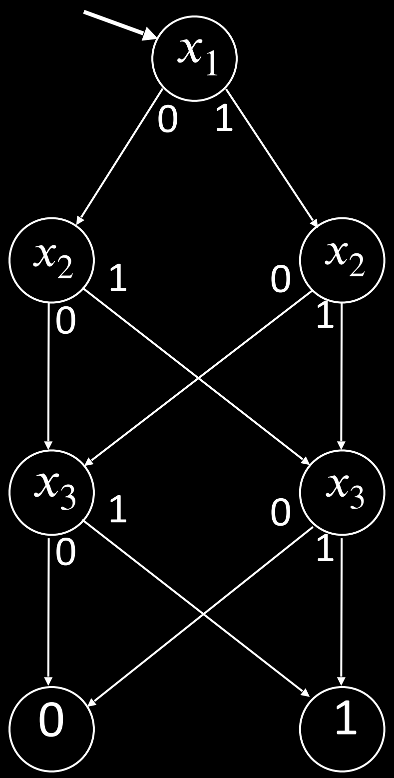




Arithmetization Method

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

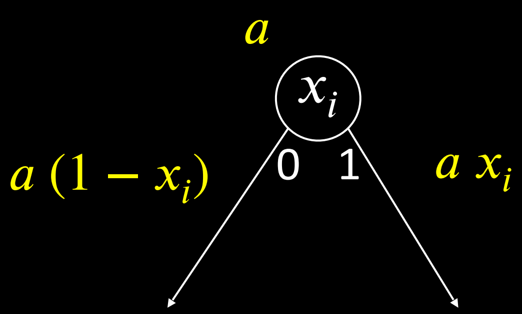
$$a \wedge b \quad \rightarrow \quad a \times b = ab$$
$$\overline{a} \quad \rightarrow \quad (1 - a)$$
$$a \vee b \quad \rightarrow \quad a + b - ab$$

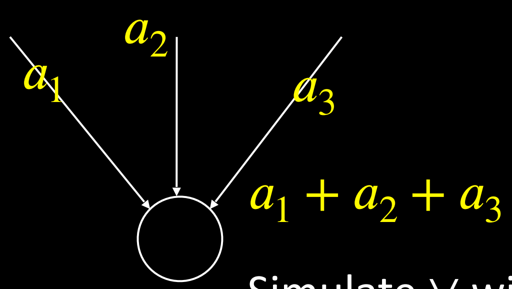


Replace Boolean labeling with arithmetical labeling

Inductive rules:

Start node labeled 1



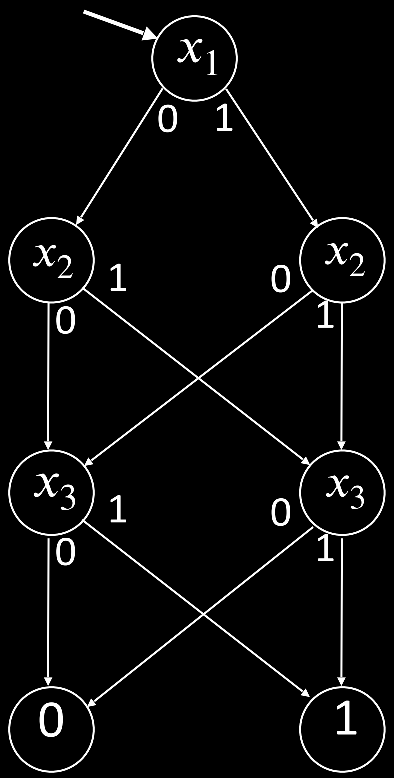


Simulate $\vee$ with $+$ because the BP is acyclic.
The execution path can enter a node
at most one time.

Arithmetization Method

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

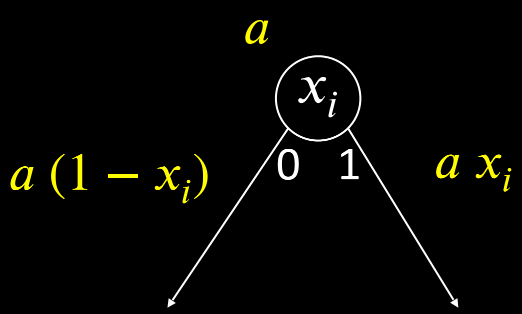
$$
\begin{array}{ccc}
a \wedge b & \rightarrow & a \times b = ab \\
\overline{a} & \rightarrow & (1 - a) \\
a \vee b & \rightarrow & a + b - ab
\end{array}
$$

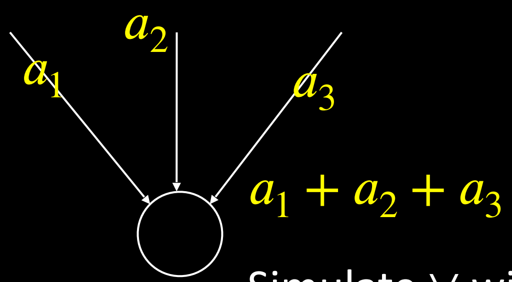


Replace Boolean labeling with arithmetical labeling

Inductive rules:

Start node labeled 1





Simulate $\vee$ 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.

Non-Boolean Labeling

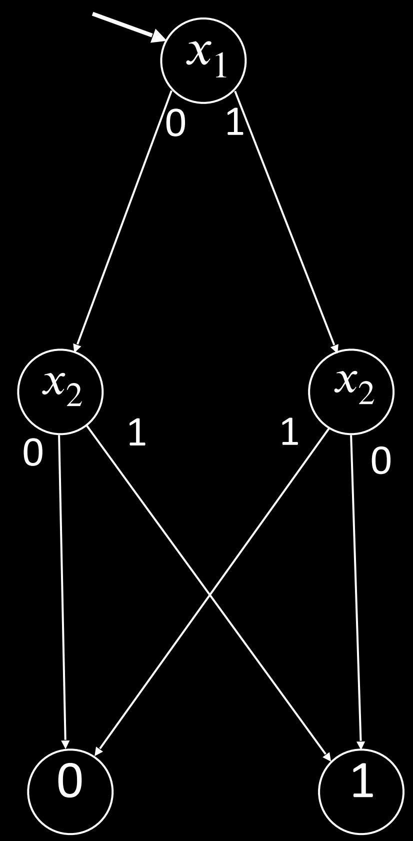
Use the arithmetized interpretation of the BP's computation to define its operation on non-Boolean inputs.

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

Non-Boolean Labeling

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

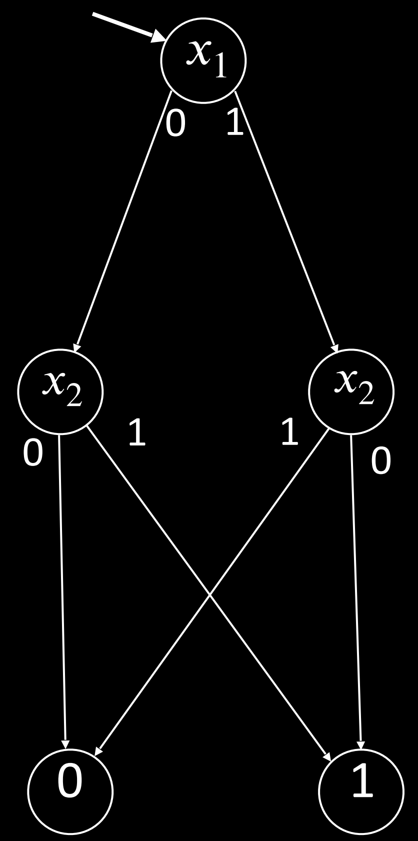
Example: $x_1 = 2, \quad x_2 = 3$



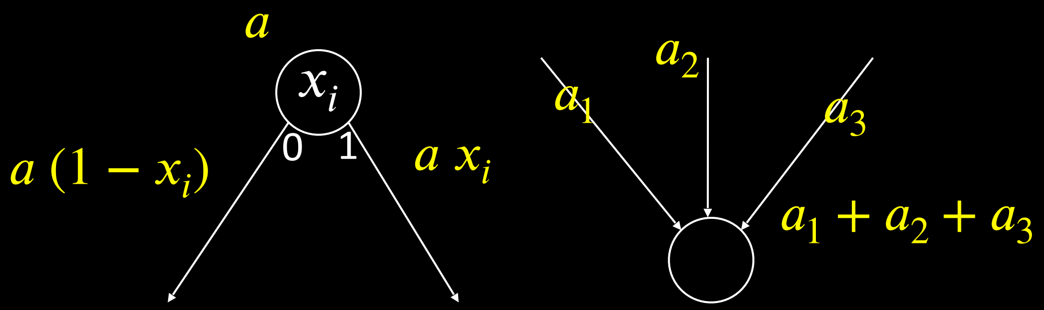
Non-Boolean Labeling

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

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



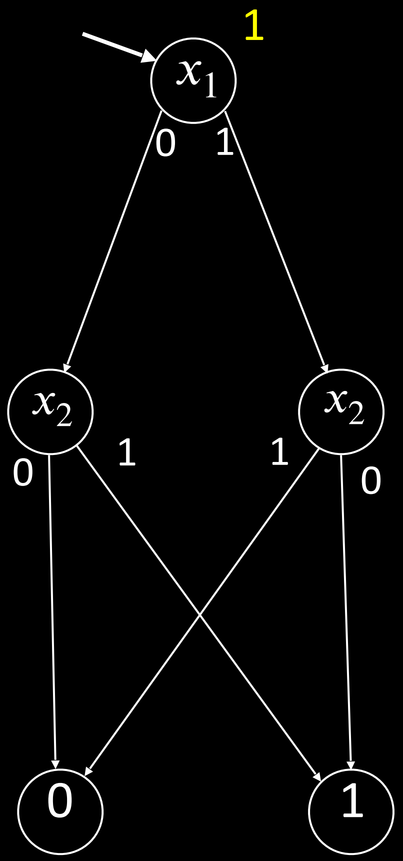
Recall labeling rules:



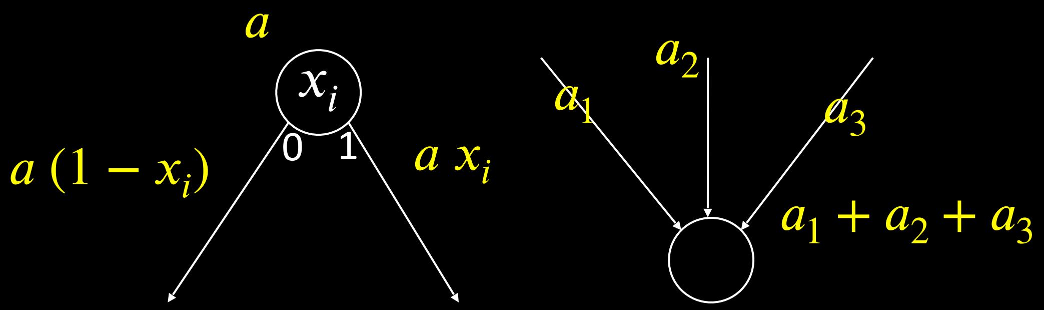
Non-Boolean Labeling

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

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



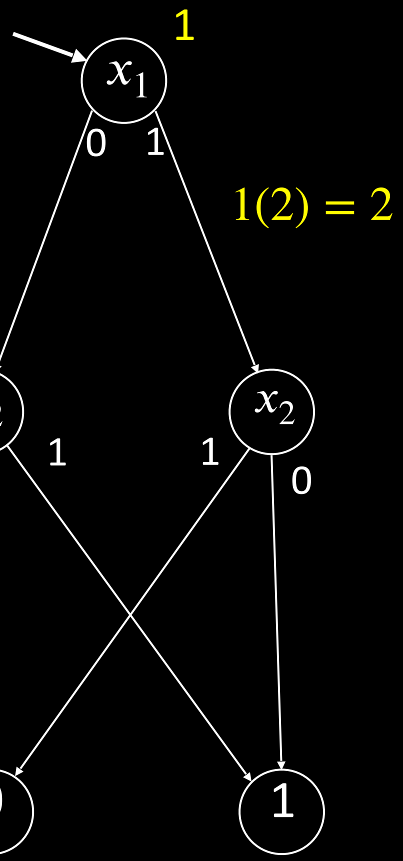
Recall labeling rules:



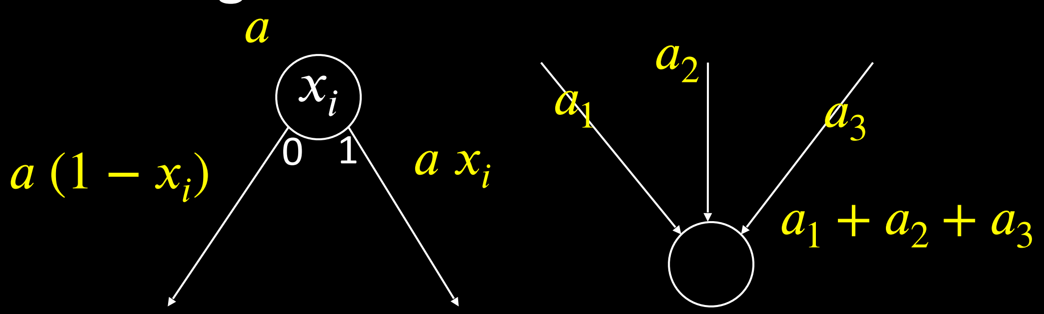
Non-Boolean Labeling

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

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



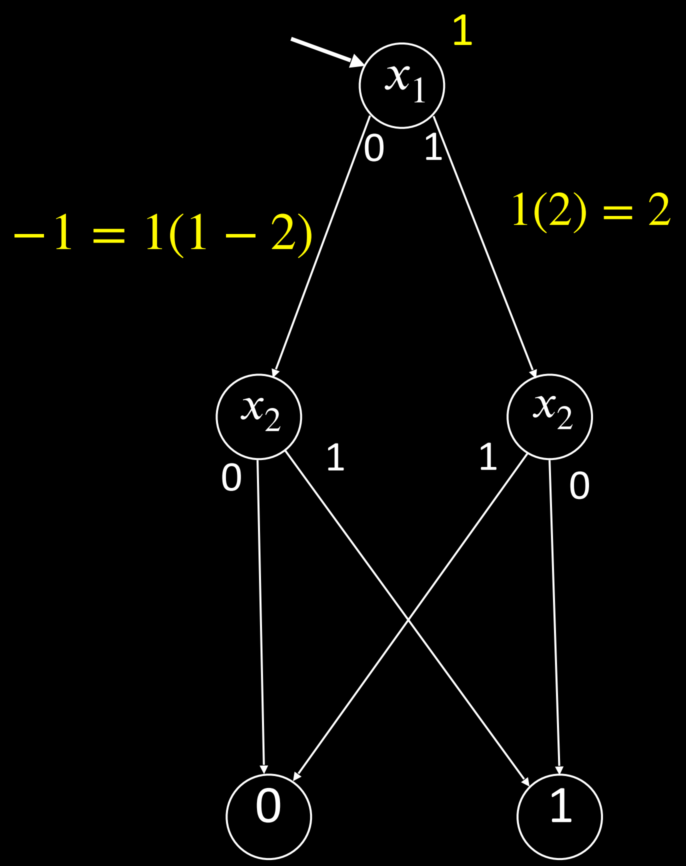
Recall labeling rules:



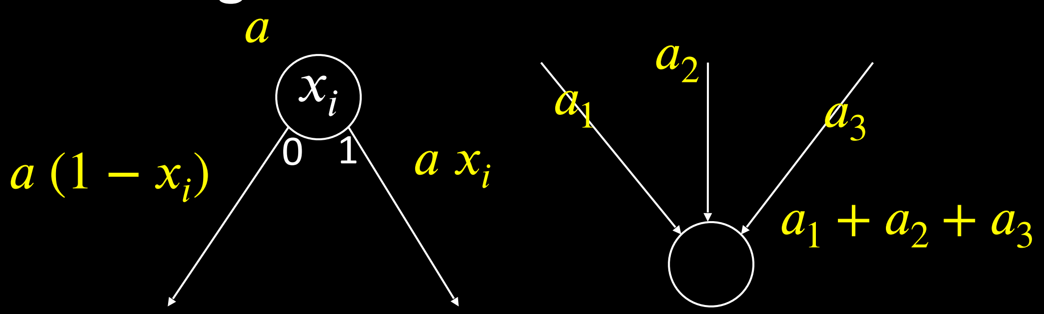
Non-Boolean Labeling

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

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



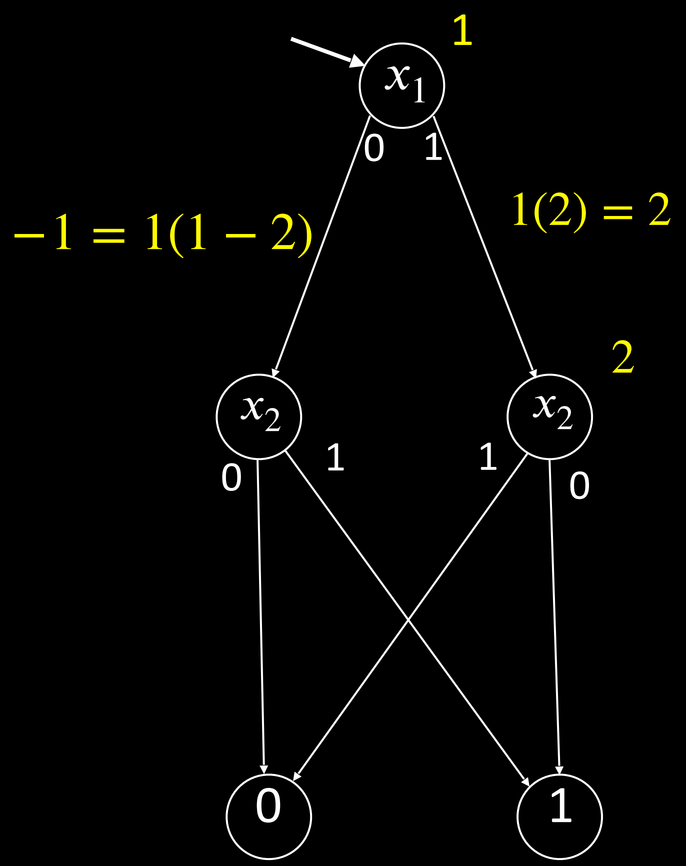
Recall labeling rules:



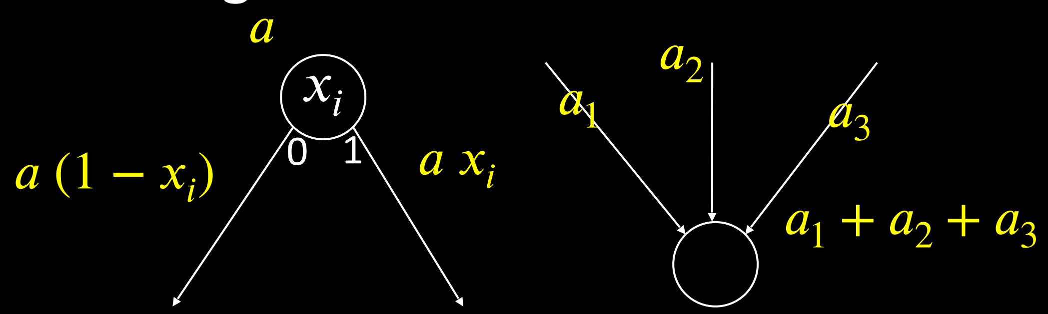
Non-Boolean Labeling

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

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



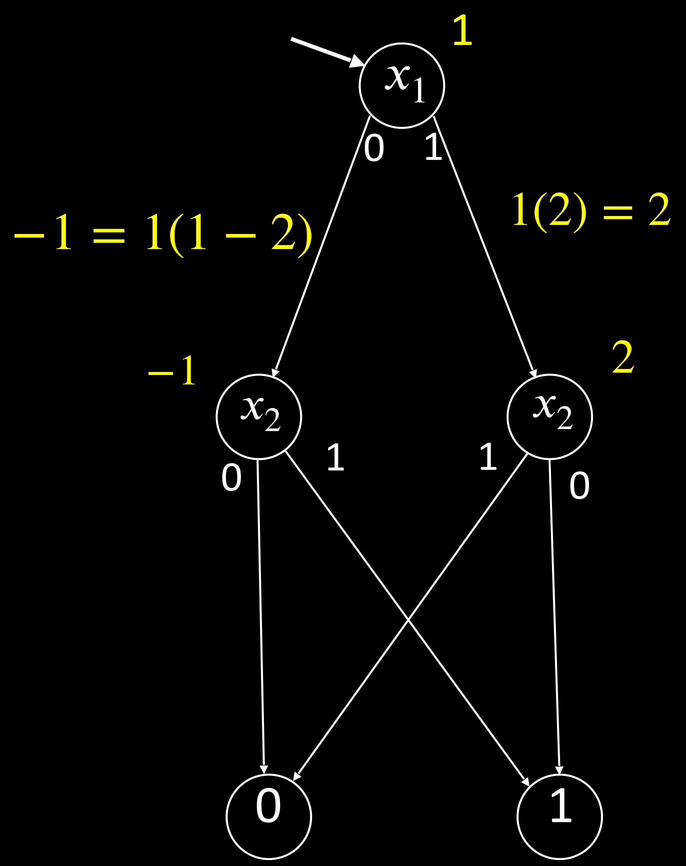
Recall labeling rules:



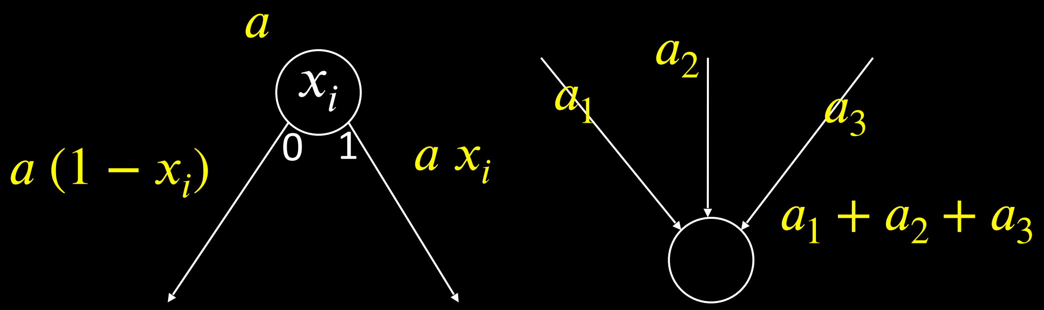
Non-Boolean Labeling

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

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



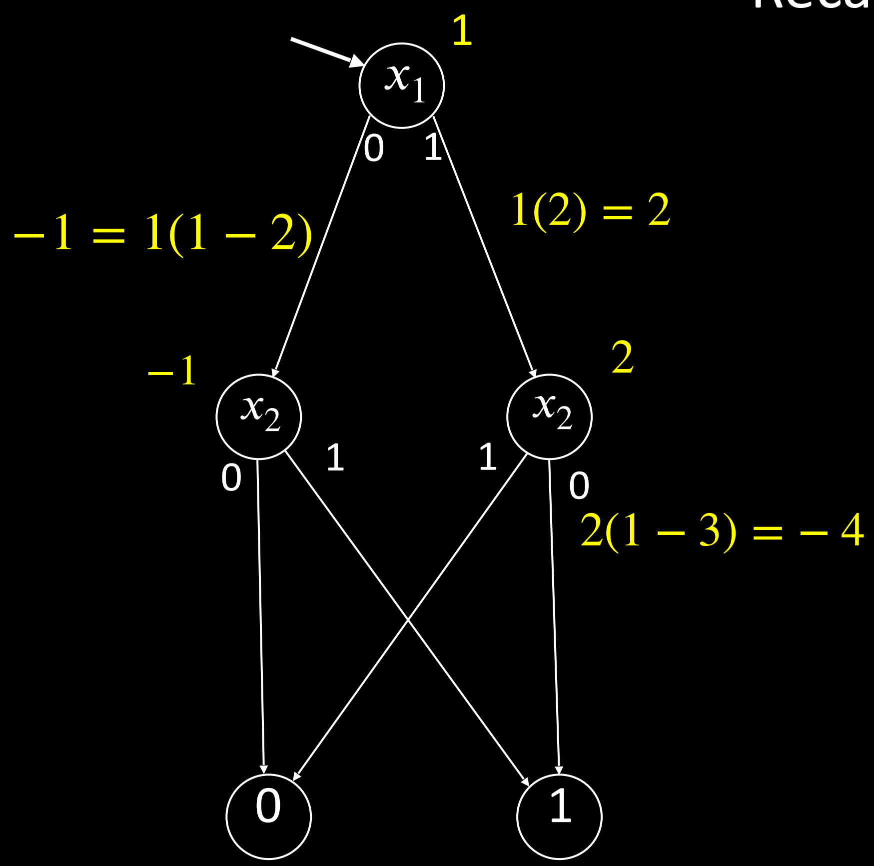
Recall labeling rules:



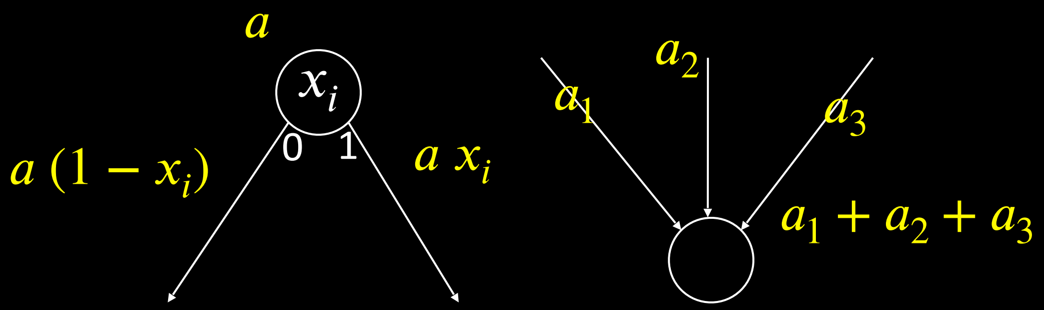
Non-Boolean Labeling

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

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



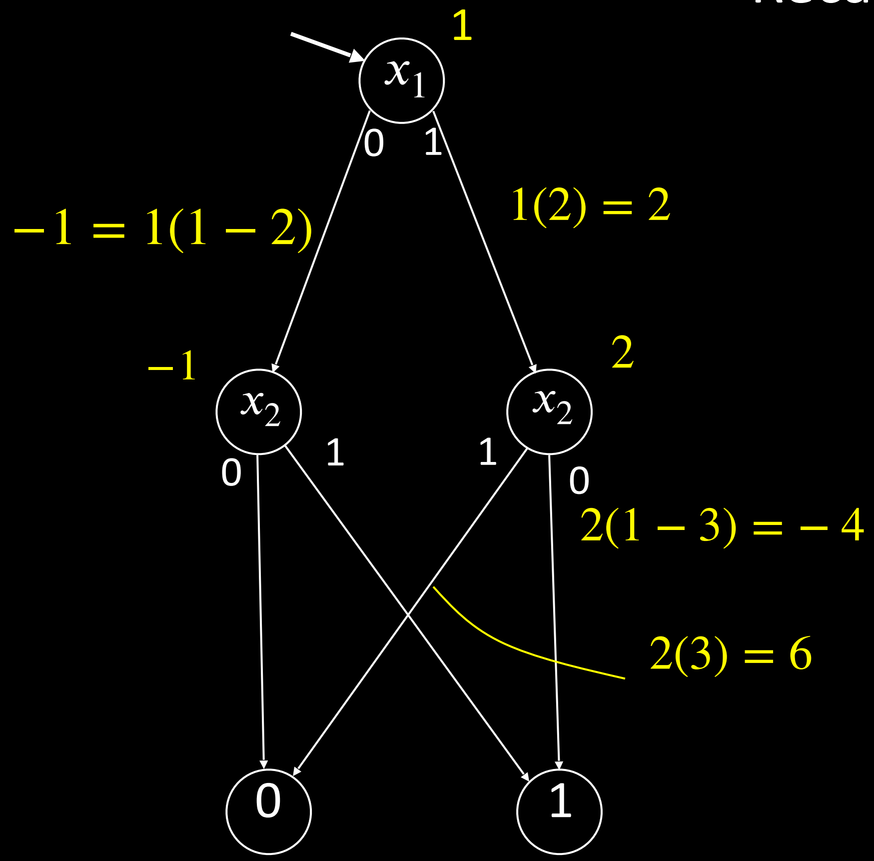
Recall labeling rules:



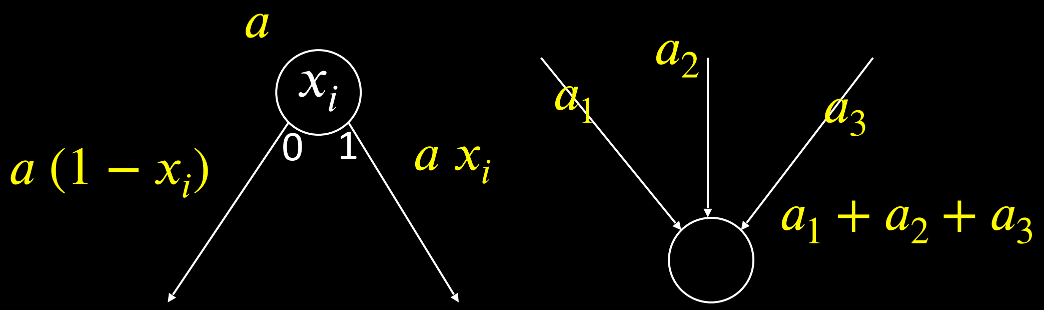
Non-Boolean Labeling

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

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



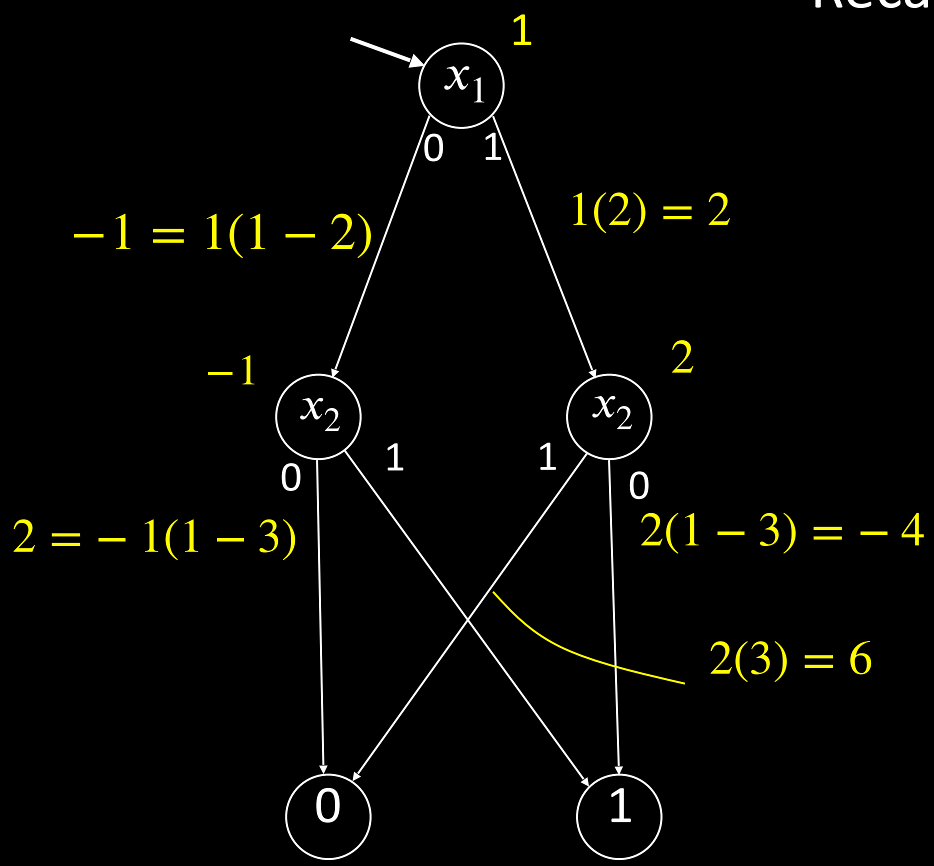
Recall labeling rules:



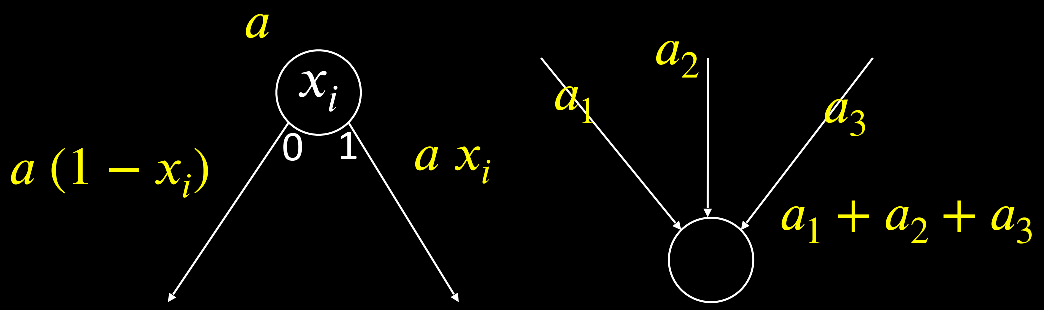
Non-Boolean Labeling

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

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



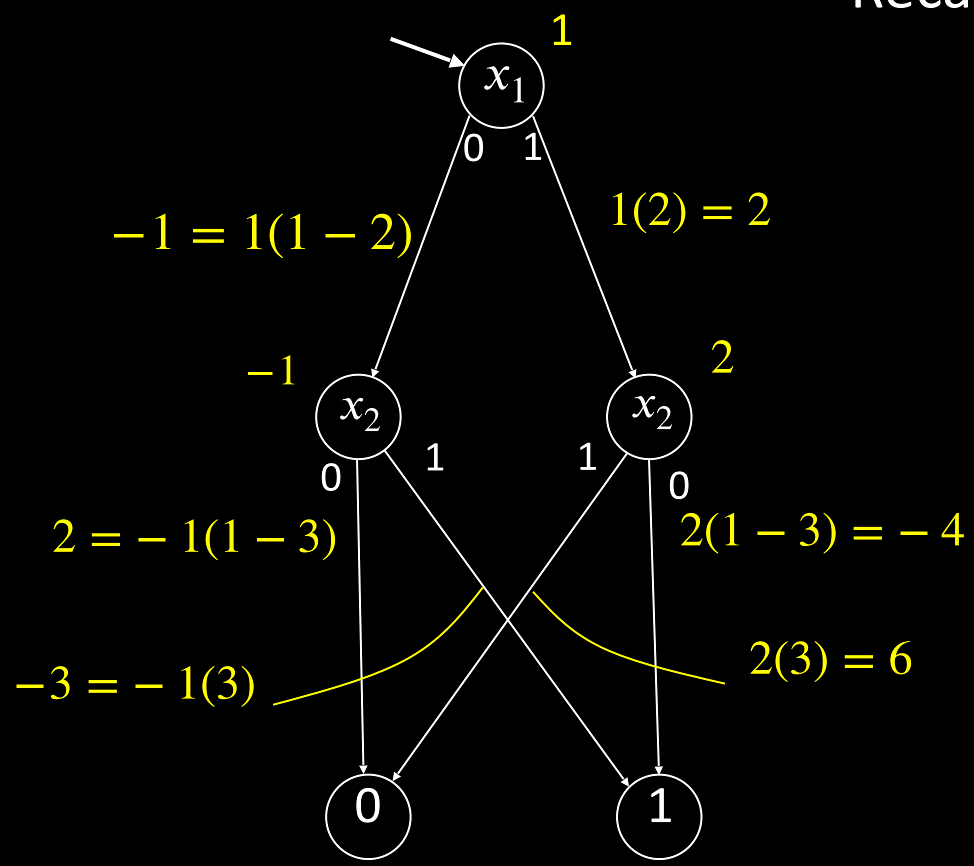
Recall labeling rules:



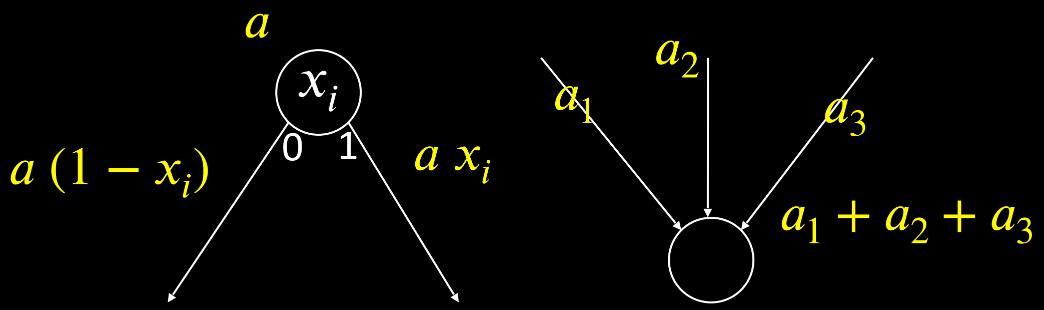
Non-Boolean Labeling

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

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



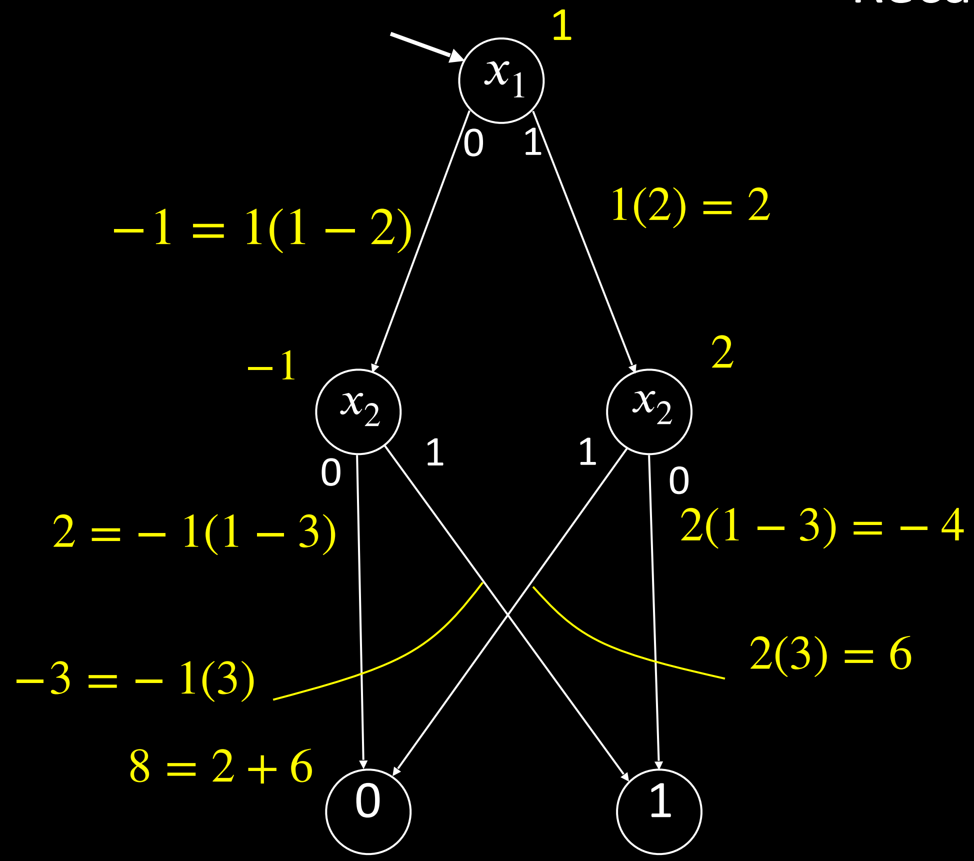
Recall labeling rules:



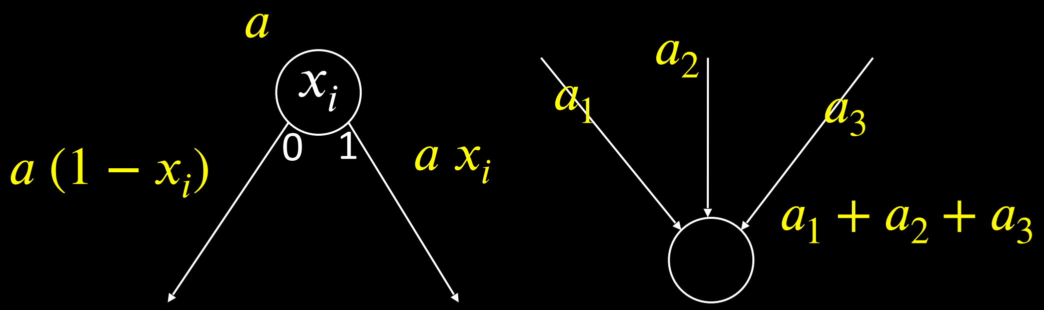
Non-Boolean Labeling

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

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



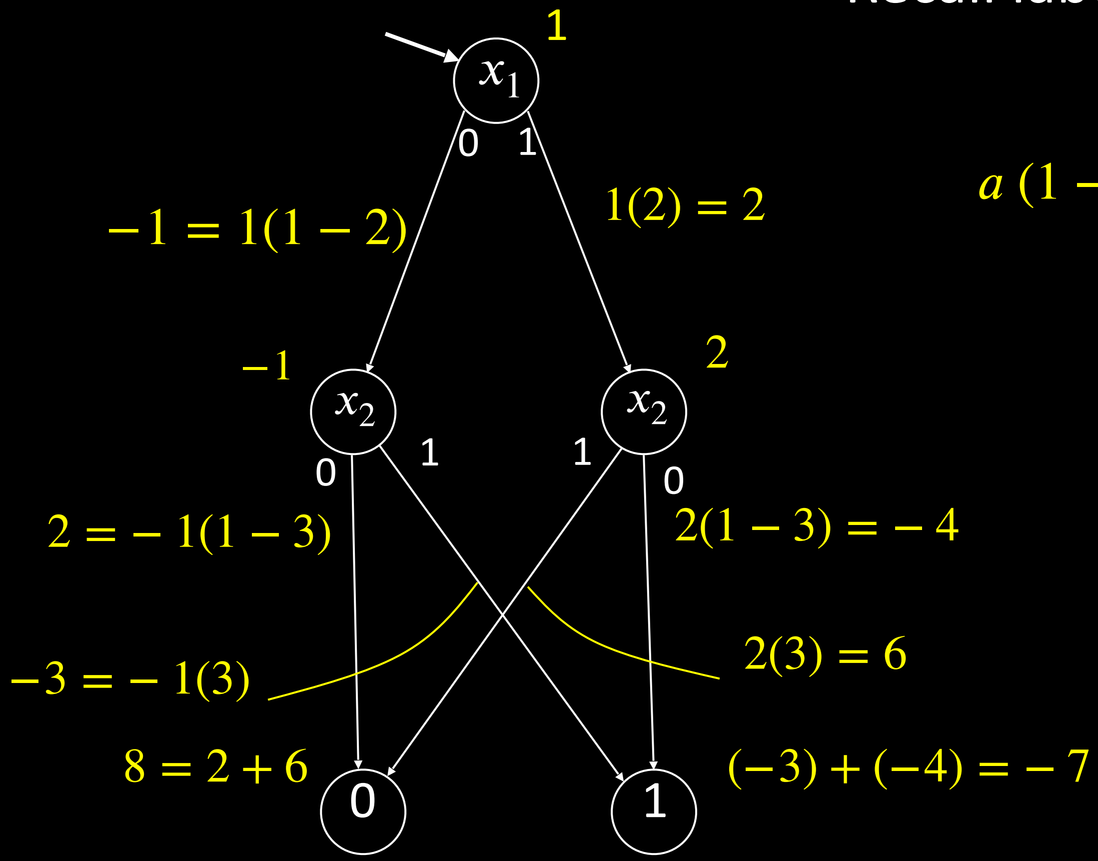
Recall labeling rules:



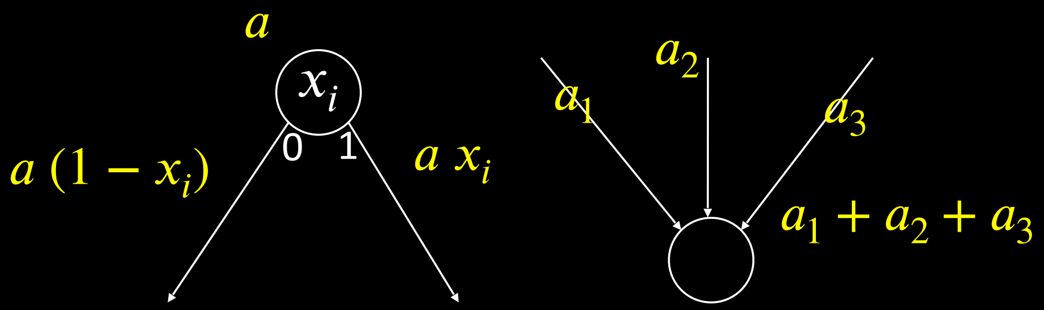
Non-Boolean Labeling

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

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



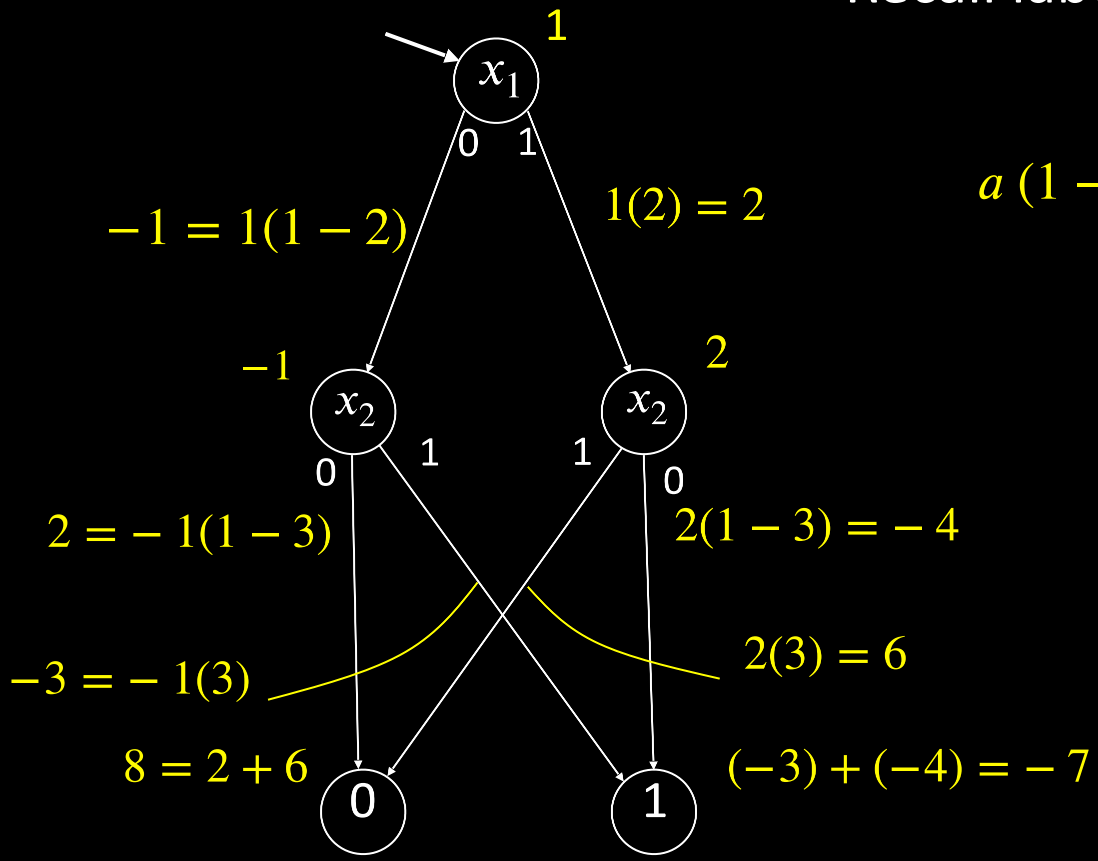
Recall labeling rules:



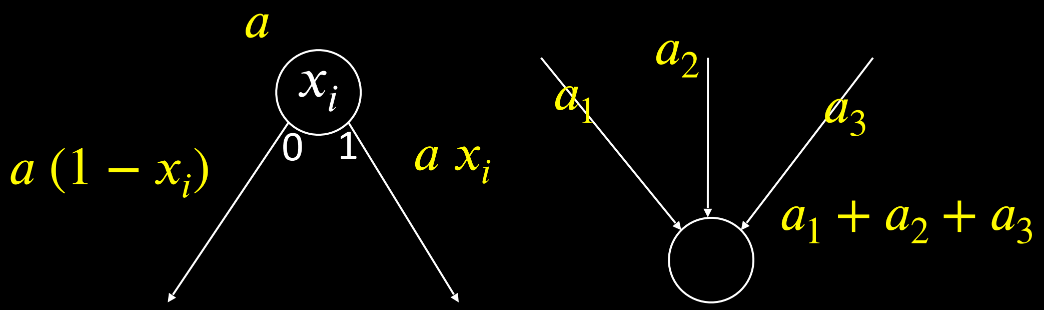
Non-Boolean Labeling

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

Example: $x_1 = 2, \quad x_2 = 3$ Output = -7



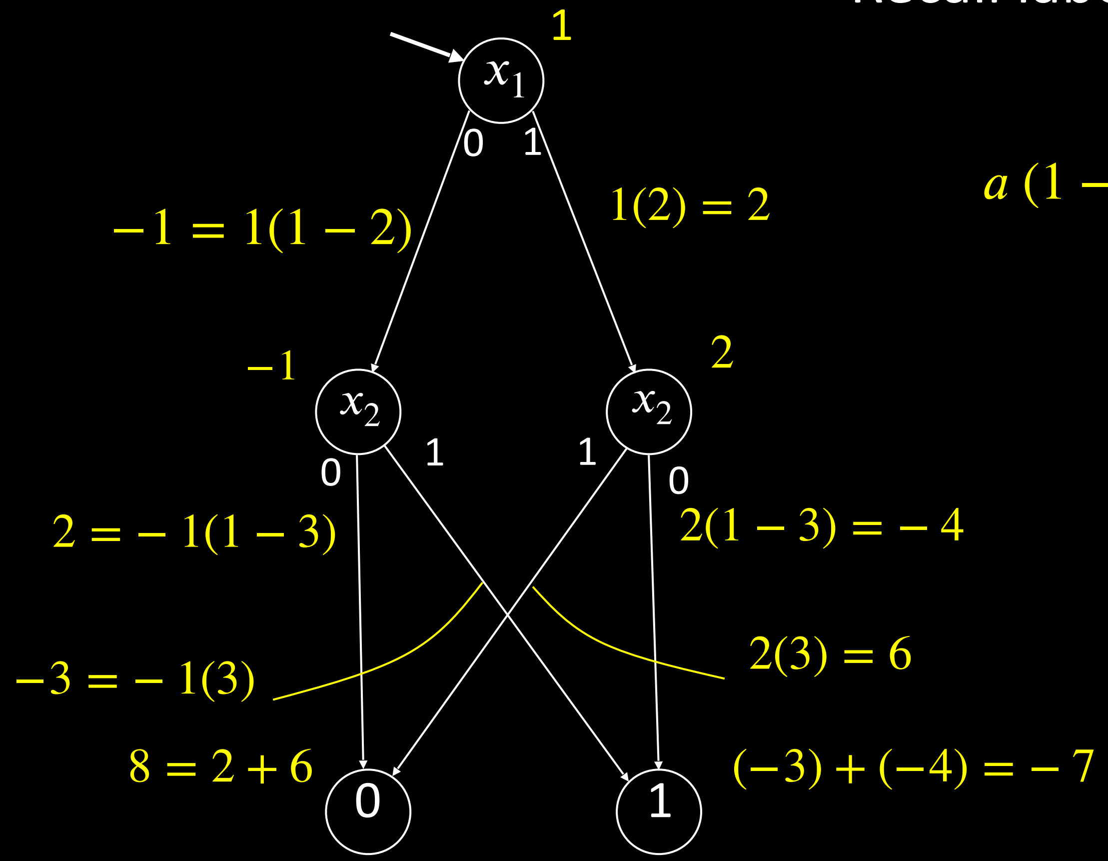
Recall labeling rules:



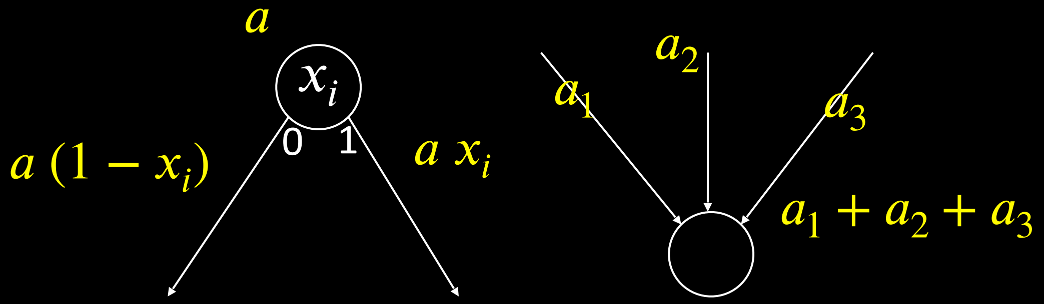
Non-Boolean Labeling

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

Example: $x_1 = 2$, $x_2 = 3$ Output = -7



Recall labeling rules:

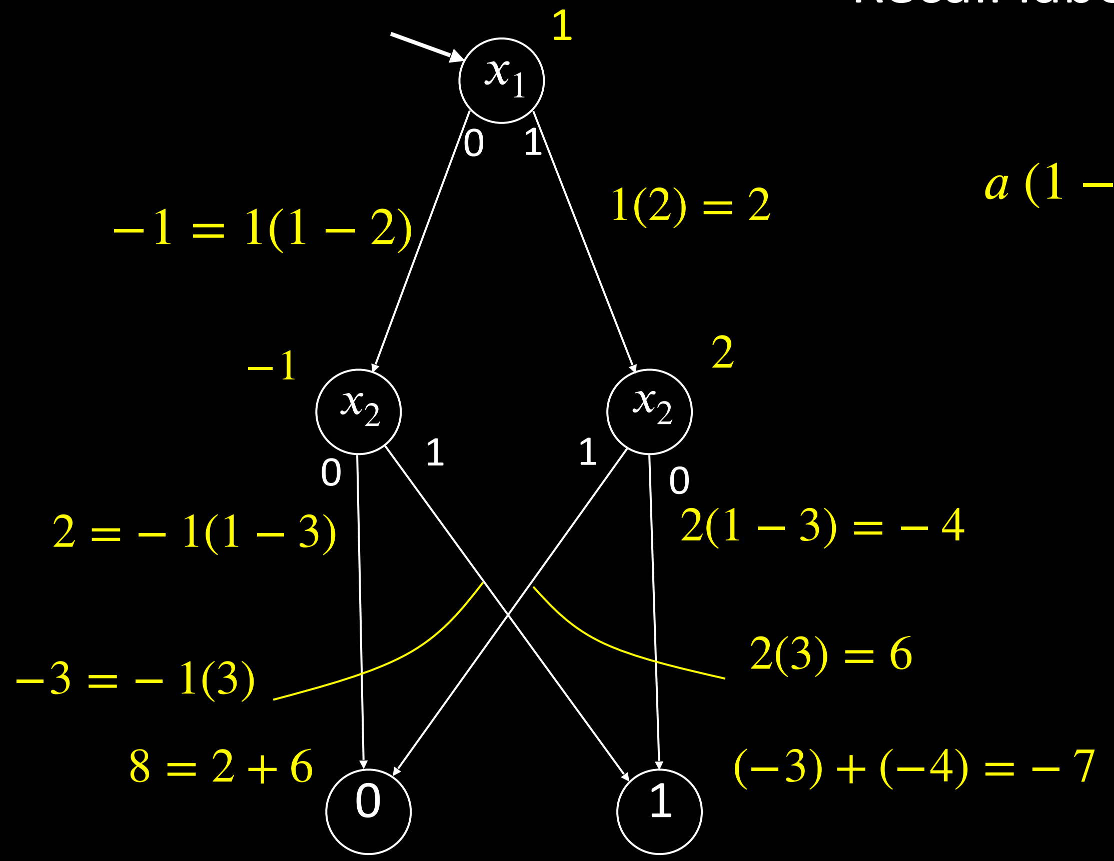


Algorithm sketch for EQROBP: “On input $\langle B_1, B_2 \rangle$

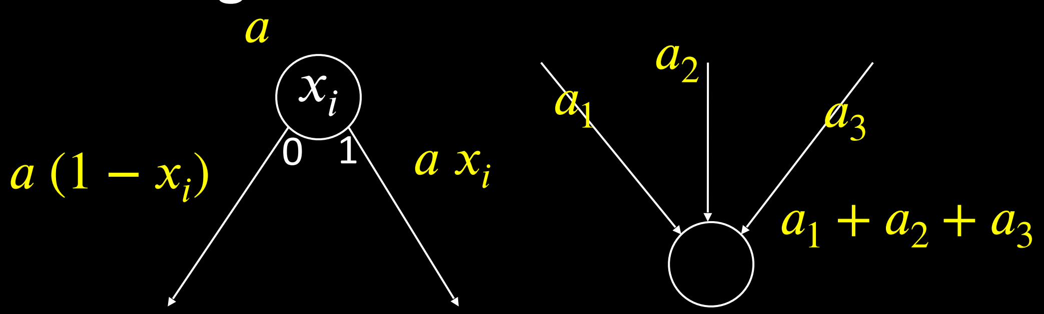
Non-Boolean Labeling

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

Example: $x_1 = 2, \quad x_2 = 3$ Output = -7



Recall labeling rules:



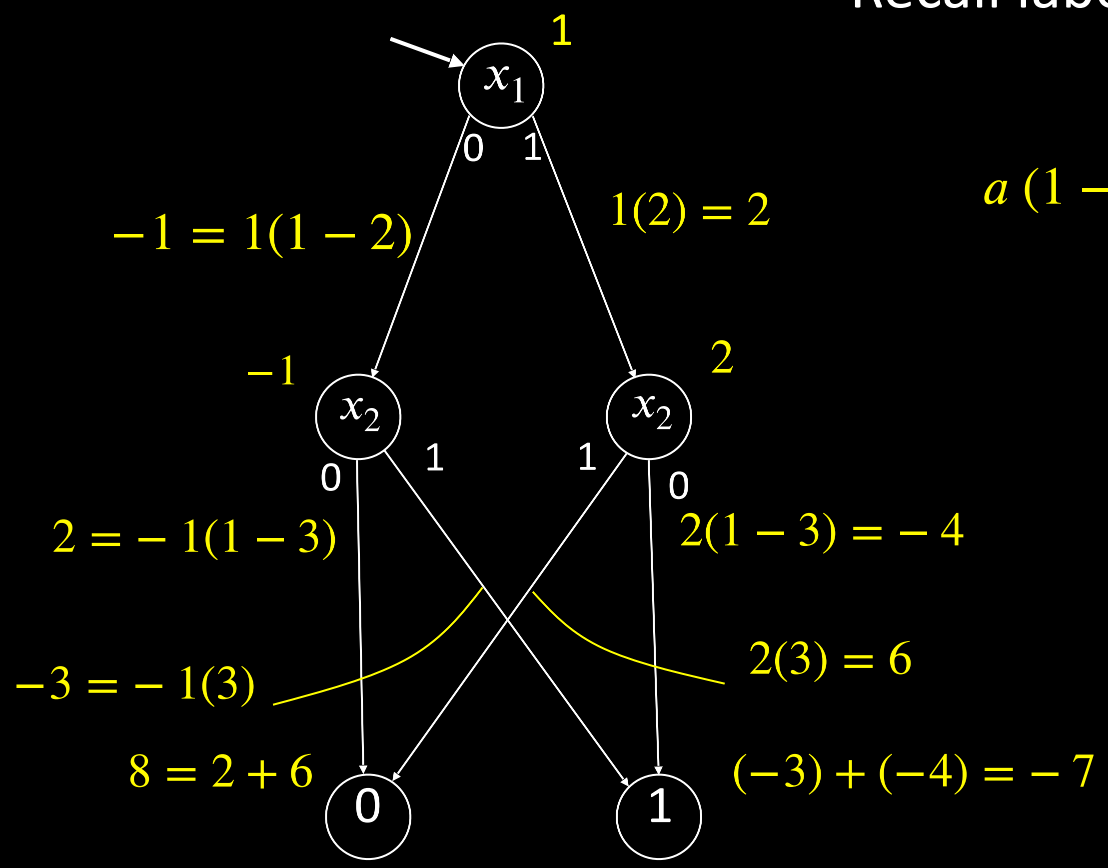
Algorithm sketch for EQROBP: "On input $\langle B_1, B_2 \rangle$

1. Pick a random *non-Boolean* input assignment.

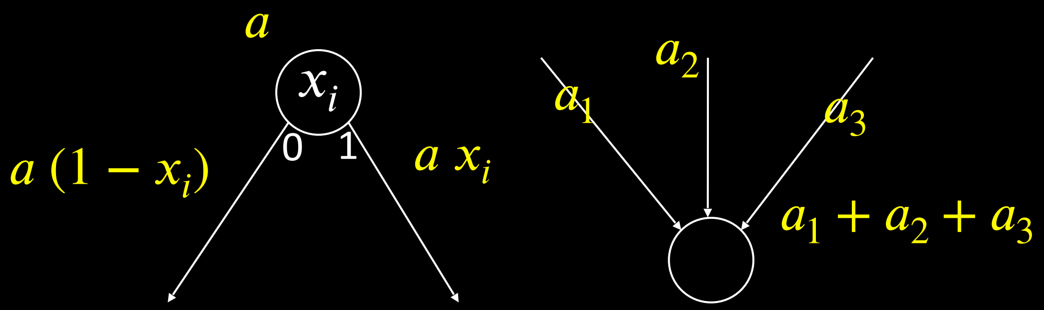
Non-Boolean Labeling

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

Example: $x_1 = 2$, $x_2 = 3$ Output = -7



Recall labeling rules:



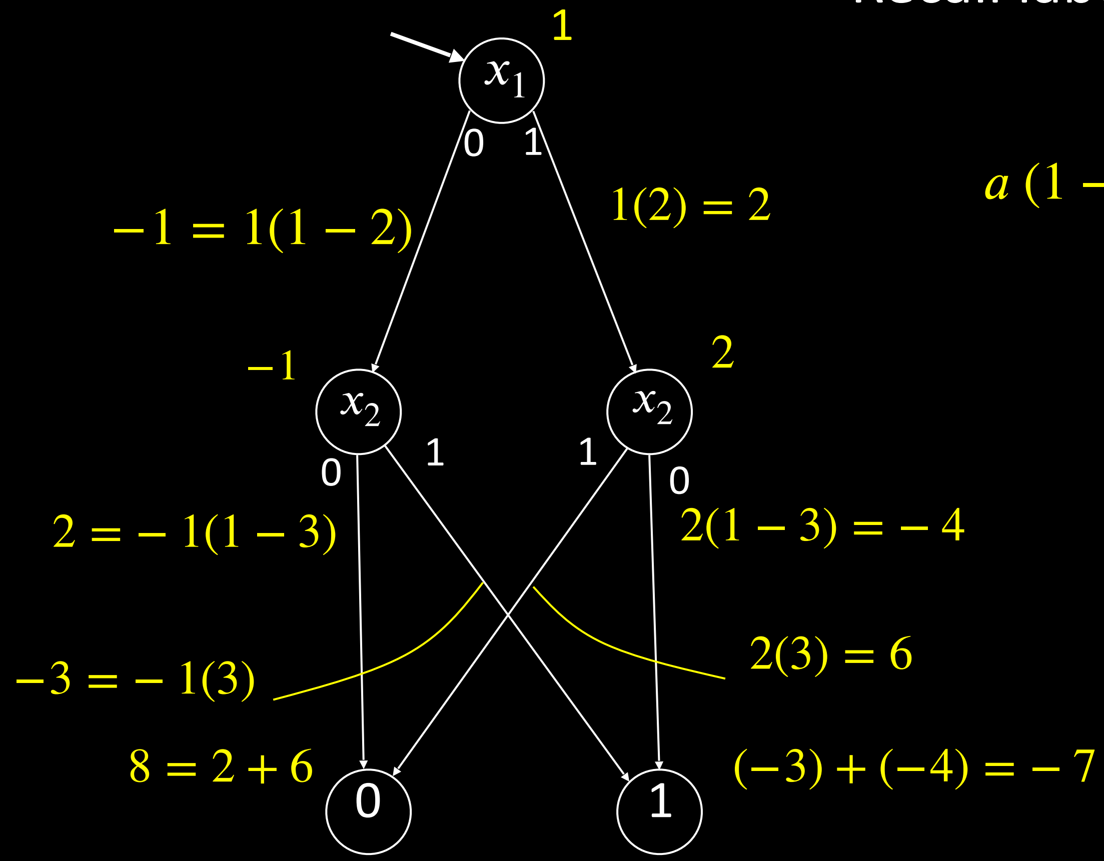
Algorithm sketch for EQROBP: "On input $\langle B_1, B_2\rangle$

1. Pick a random *non-Boolean* input assignment.
2. Evaluate B_1 and B_2 on that assignment.

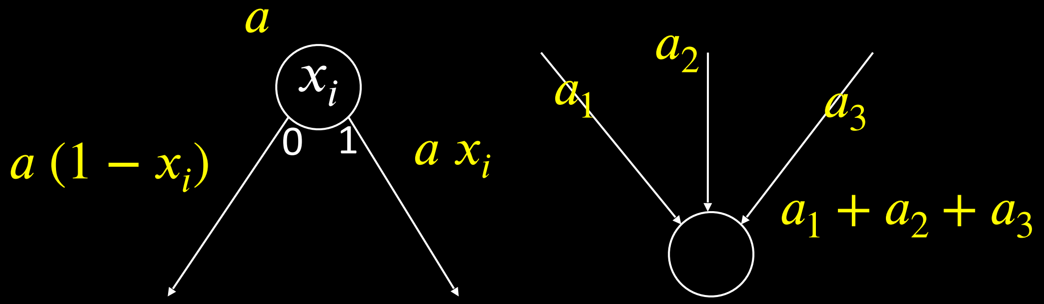
Non-Boolean Labeling

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

Example: $x_1 = 2$, $x_2 = 3$ Output = -7



Recall labeling rules:



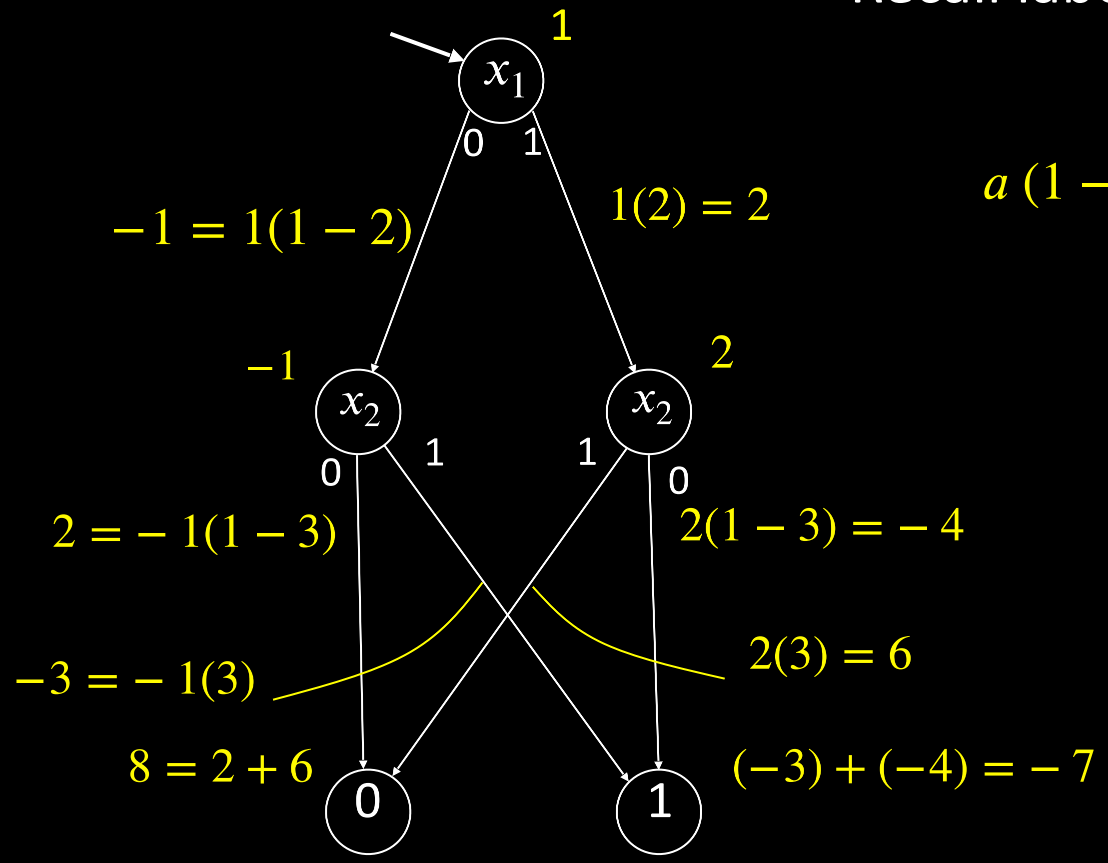
Algorithm sketch for EQ_{ROBP}: "On input $\langle B_1, B_2 \rangle$

1. Pick a random *non-Boolean* input assignment.
2. Evaluate B_1 and B_2 on that assignment.
3. If B_1 and B_2 disagree then *reject*.
   If they agree then *accept*."

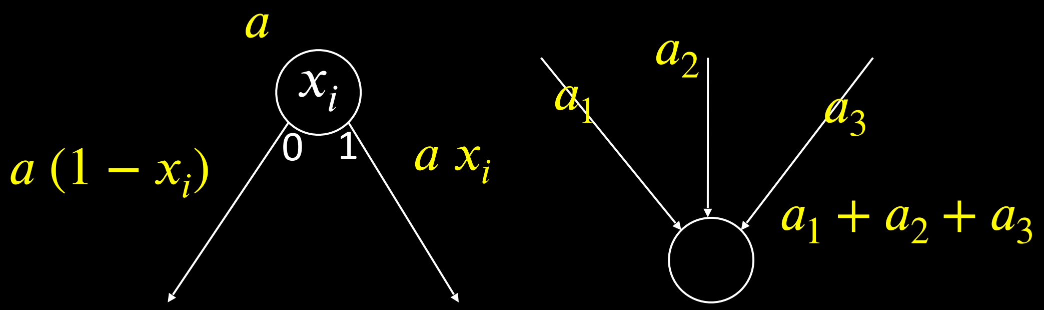
Non-Boolean Labeling

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

Example: $x_1 = 2,\ \ x_2 = 3$ Output = -7



Recall labeling rules:



Algorithm sketch for EQROBP: "On input $\langle B_1, B_2 \rangle$

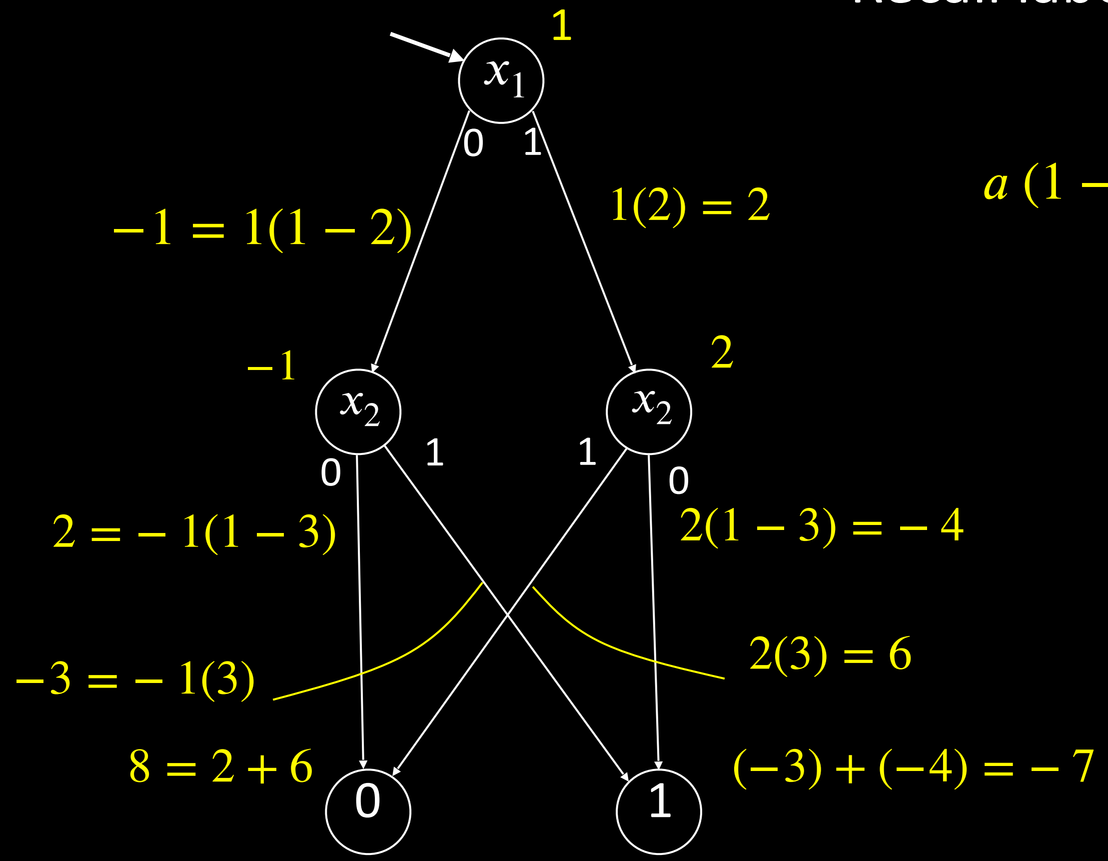
1. Pick a random *non-Boolean* input assignment.
2. Evaluate B_1 and B_2 on that assignment.
3. If B_1 and B_2 disagree then *reject*.
   If they agree then *accept*."

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

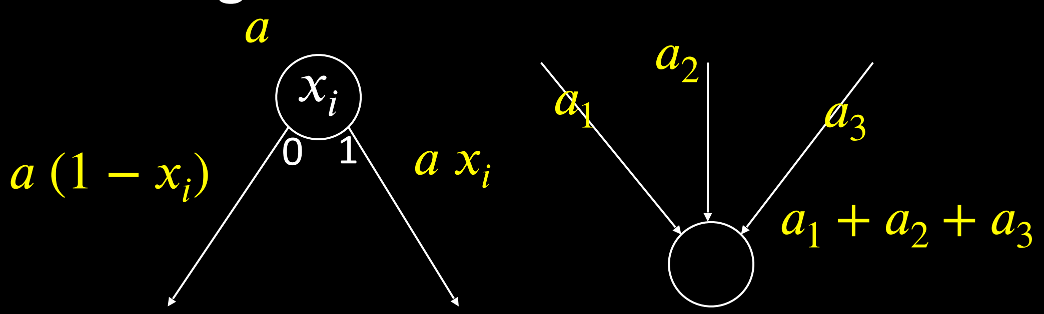
Non-Boolean Labeling

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

Example: $x_1 = 2$, $x_2 = 3$ Output = -7



Recall labeling rules:

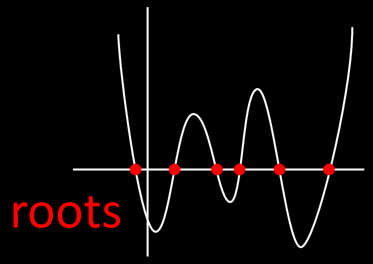


Algorithm sketch for EQROBP: "On input $\langle B_1, B_2 \rangle$

1. Pick a random *non-Boolean* input assignment.
2. Evaluate B_1 and B_2 on that assignment.
3. If B_1 and B_2 disagree then *reject*.
   If they agree then *accept*."

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

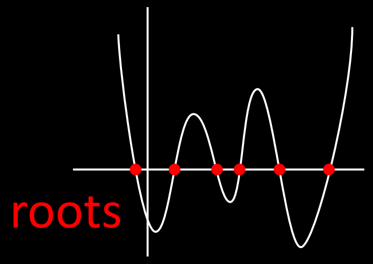
Roots of Polynomials



Let $p(x) = a_0 x^d + a_1 x^{d-1} + a_2 x^{d-2} + \cdots + a_d$ be a polynomial.

Corollary 2: If $p(x) \neq 0$ has degree $\leq d$ and we pick a random $r \in \mathbb{F}_q$, then $\Pr\left[\, p(r) = 0 \,\right] \leq \frac{d}{q}$.

Roots of Polynomials



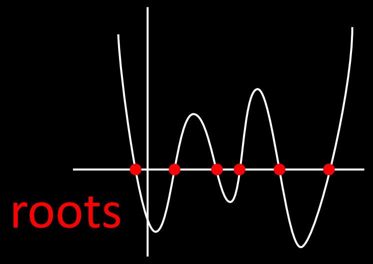
Let $p(x) = a_0 x^d + a_1 x^{d-1} + a_2 x^{d-2} + \cdots + a_d$ be a polynomial.

Corollary 2: If $p(x) \neq 0$ has degree $\leq d$ and we pick a random $r \in \mathbb{F}_q$, then Pr$\left[\, p(r) = 0 \,\right] \leq \frac{d}{q}$.

Proof: There are at most d roots out of q possibilities.

Theorem (Schwartz-Zippel): If $p\left(x_1, \ldots, x_m\right) \neq 0$ has degree $\leq d$ in each x_i and

Roots of Polynomials



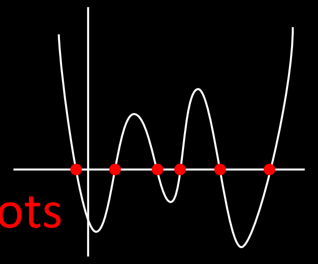
Let $p(x) = a_0 x^d + a_1 x^{d-1} + a_2 x^{d-2} + \cdots + a_d$ be a polynomial.

Corollary 2: If $p(x) \neq 0$ has degree $\leq d$ and we pick a random $r \in \mathbb{F}_q$, then $\Pr\left[\, p(r) = 0 \,\right] \leq \dfrac{d}{q}$.

Proof: There are at most d roots out of q possibilities.

Theorem (Schwartz-Zippel): If $p\left(x_1, \ \ldots, \ x_m\right) \neq 0$ has degree $\leq d$ in each x_i and we pick random $r_1, \ldots, r_m \in \mathbb{F}_q$ then $\Pr\left[\, p\left(r_1, \ldots, r_m\right) = 0 \,\right] \leq \dfrac{md}{q}$

Roots of Polynomials



Let $p(x) = a_0 x^d + a_1 x^{d-1} + a_2 x^{d-2} + \cdots + a_d$ be a polynomial.

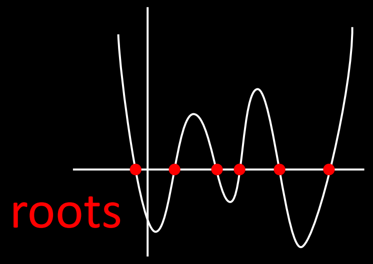
Corollary 2: If $p(x) \neq 0$ has degree $\leq d$ and we pick a random $r \in \mathbb{F}_q$, then $\Pr\left[\, p(r) = 0 \,\right] \leq \frac{d}{q}$.

Proof: There are at most d roots out of q possibilities.

Theorem (Schwartz-Zippel): If $p(x_1, \ldots, x_m) \neq 0$ has degree $\leq d$ in each x_i and we pick random $r_1, \ldots, r_m \in \mathbb{F}_q$ then $\Pr\left[\, p(r_1, \ldots, r_m) = 0 \,\right] \leq \frac{md}{q}$

Proof by induction (see text).

Roots of Polynomials



Let $p(x) = a_0 x^d + a_1 x^{d-1} + a_2 x^{d-2} + \cdots + a_d$ be a polynomial.

Corollary 2: If $p(x) \neq 0$ has degree $\leq d$ and we pick a random $r \in \mathbb{F}_q$, then $\Pr\left[\, p(r) = 0 \,\right] \leq \frac{d}{q}$.

Proof: There are at most d roots out of q possibilities.

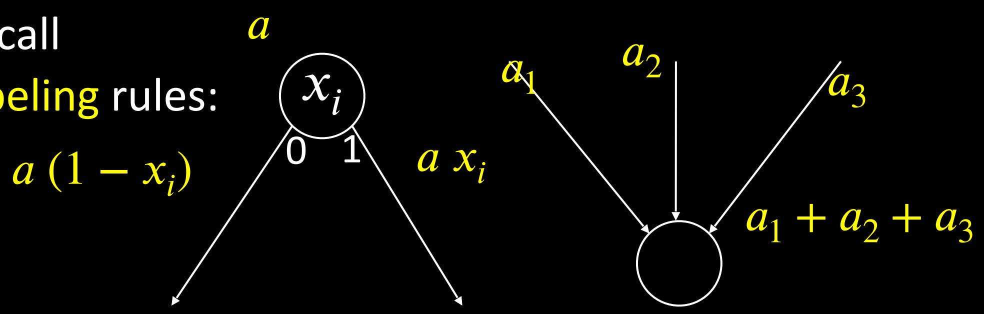
Theorem (Schwartz-Zippel): If $p\left(x_1, \ldots, x_m\right) \neq 0$ has degree $\leq d$ in each x_i and we pick random $r_1, \ldots, r_m \in \mathbb{F}_q$ then $\Pr\left[\, p\left(r_1, \ldots, r_m\right) = 0 \,\right] \leq \frac{md}{q}$

Proof by induction (see text).

Symbolic Execution

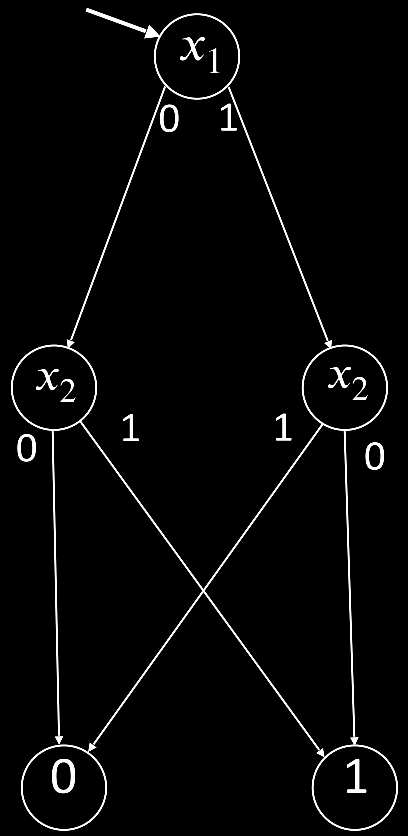
Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

Recall labeling rules:

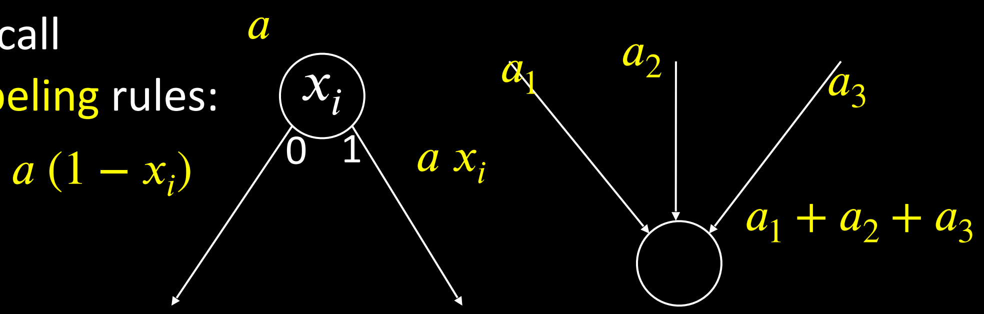


Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

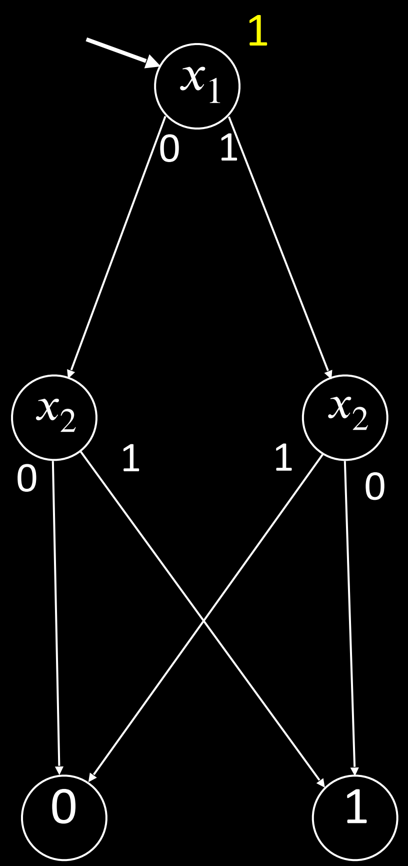


Recall labeling rules:

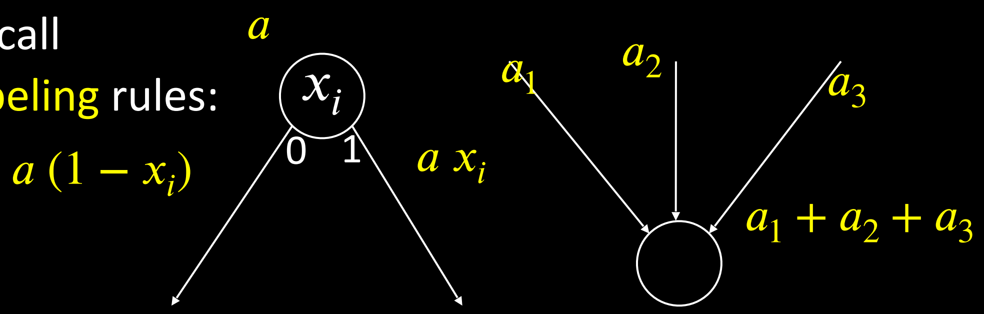


Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

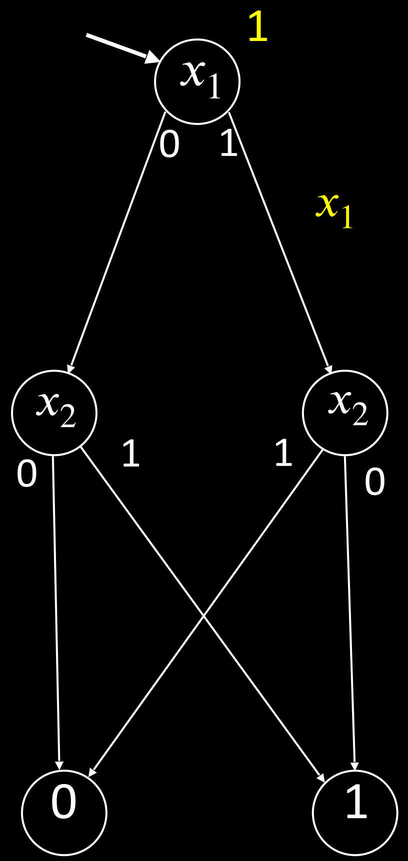


Recall
labeling rules:

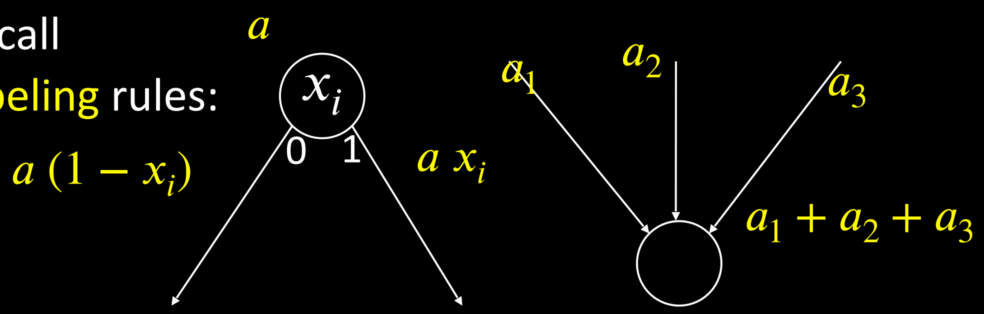


Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

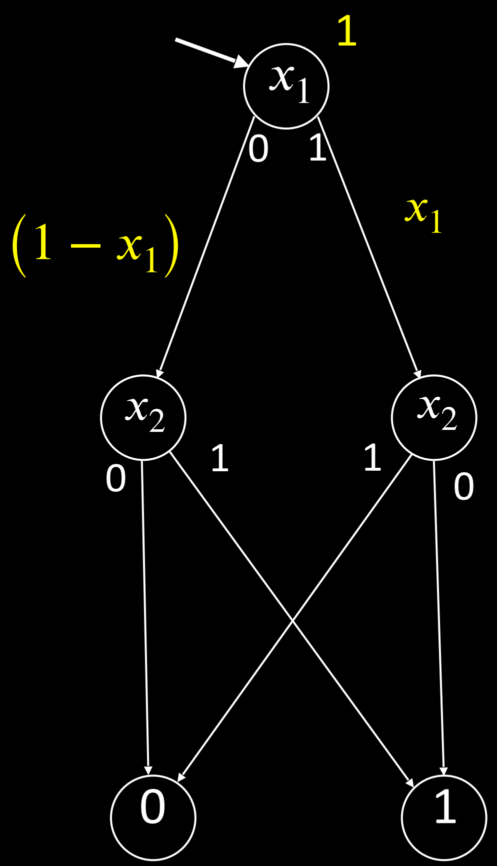


Recall labeling rules:



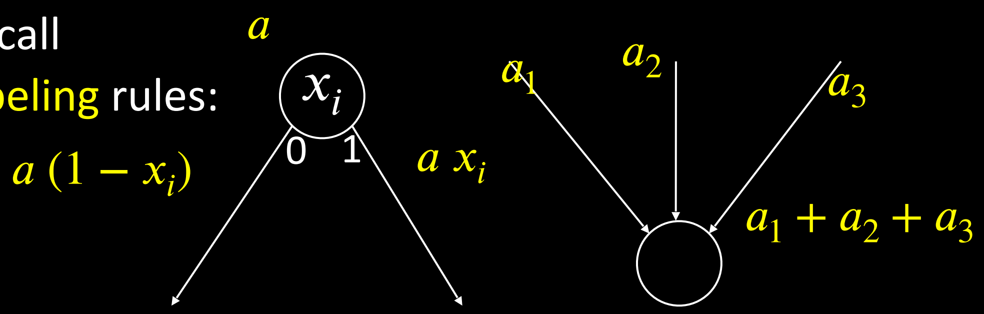
Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.



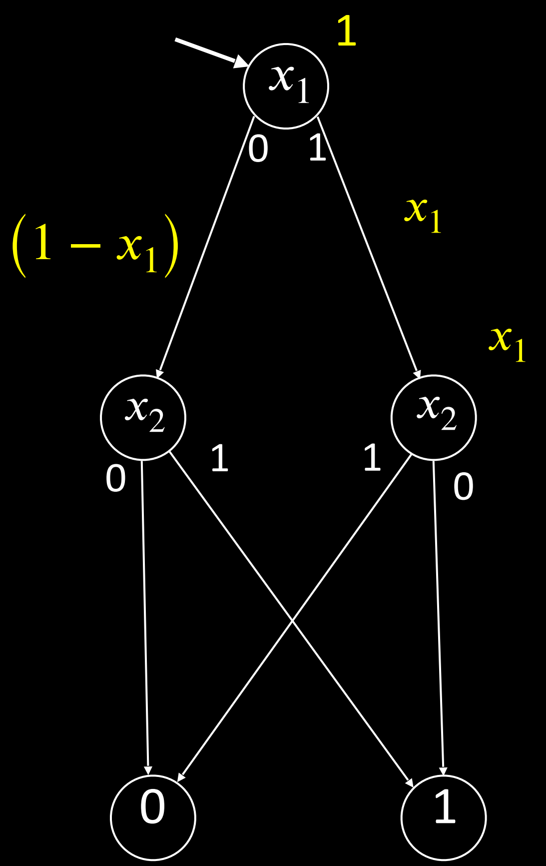
Recall

labeling rules:

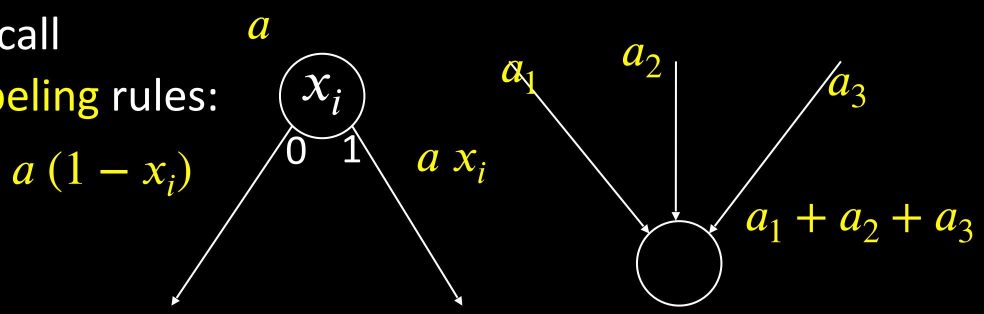


Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

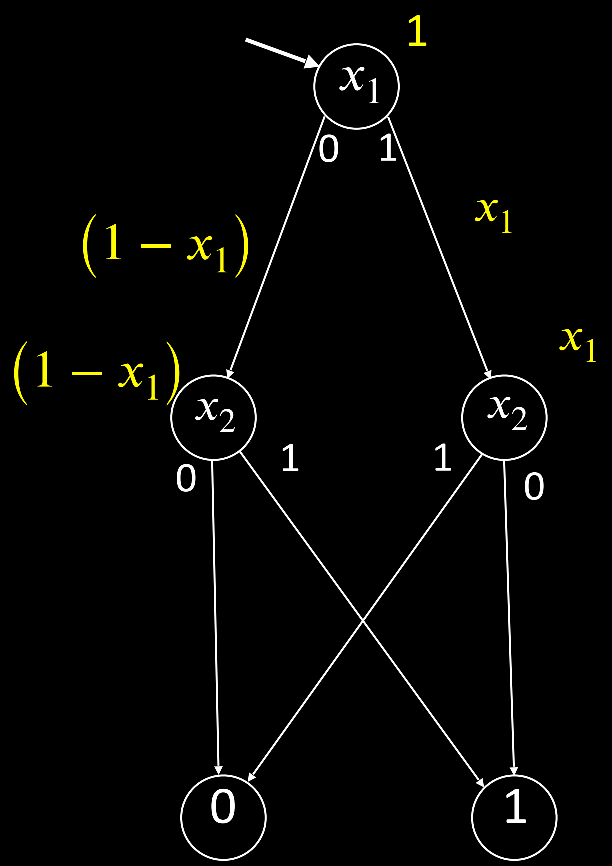


Recall labeling rules:

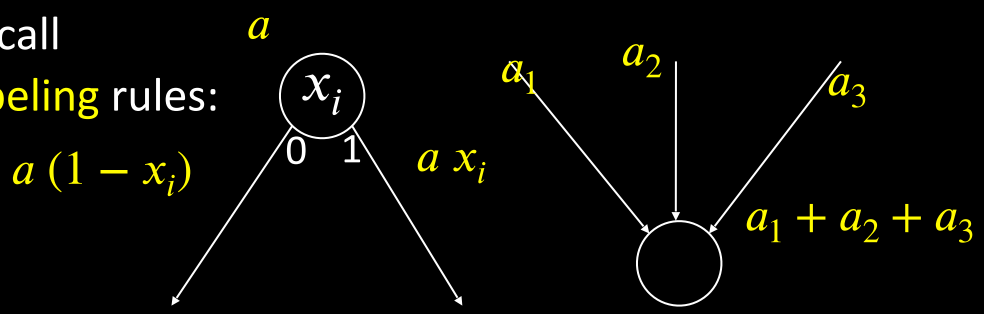


Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

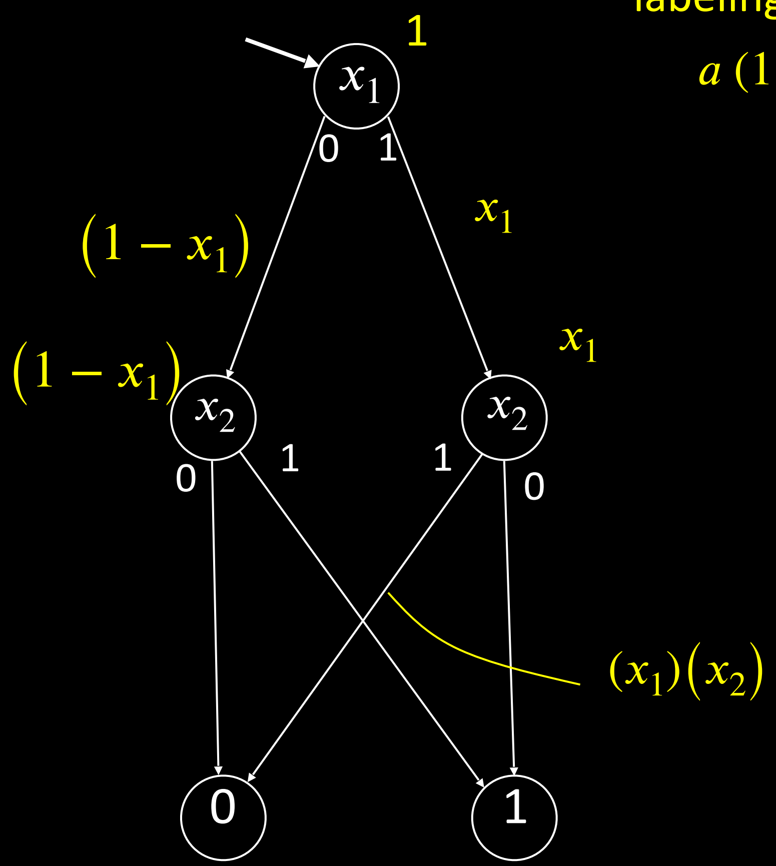


Recall
labeling rules:

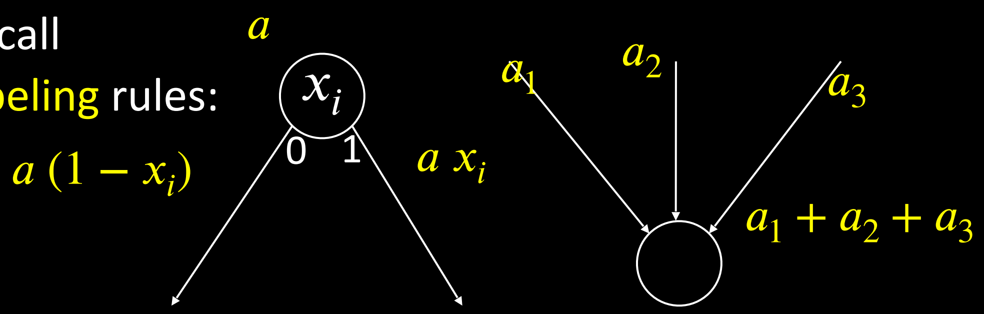


Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

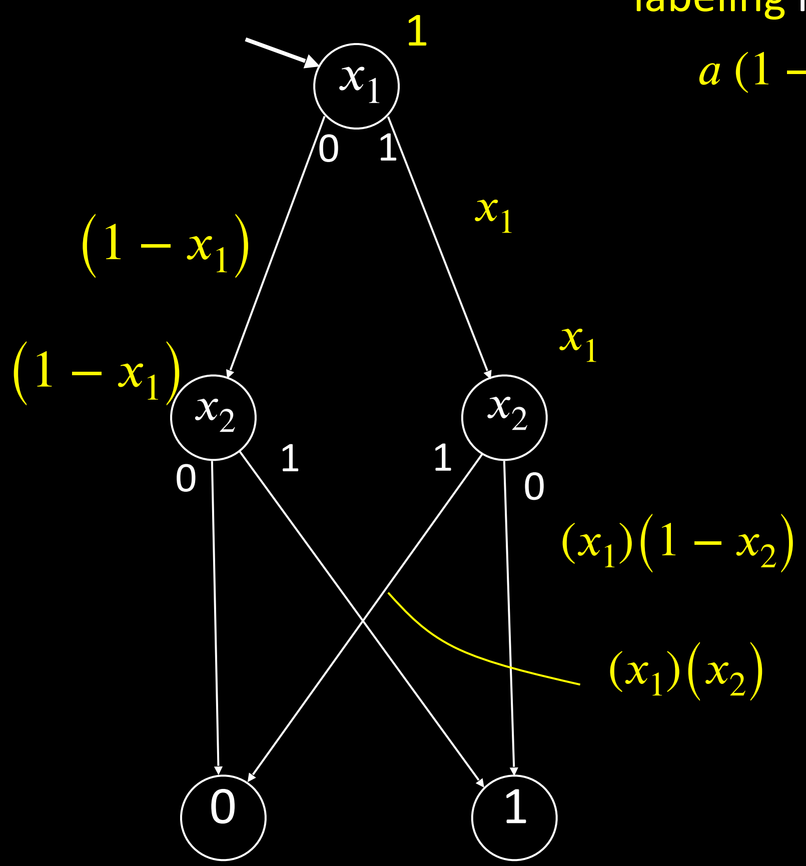


Recall labeling rules:

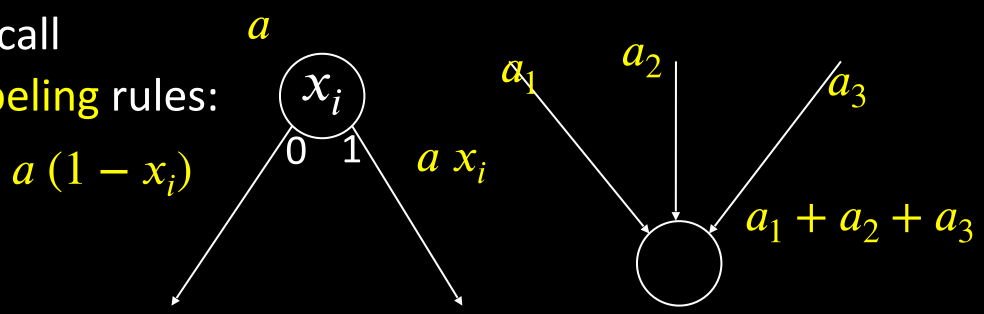


Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

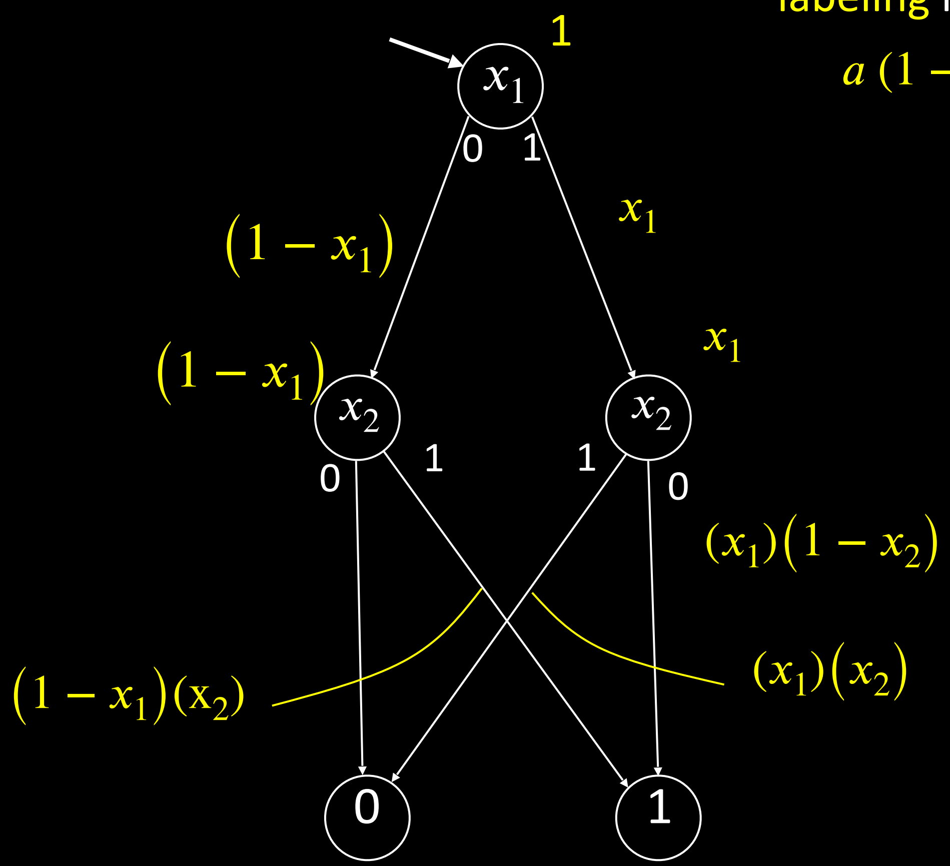


Recall
labeling rules:

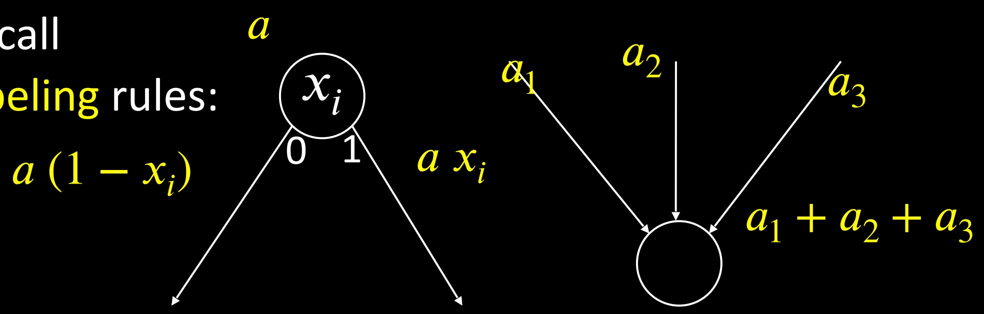


Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

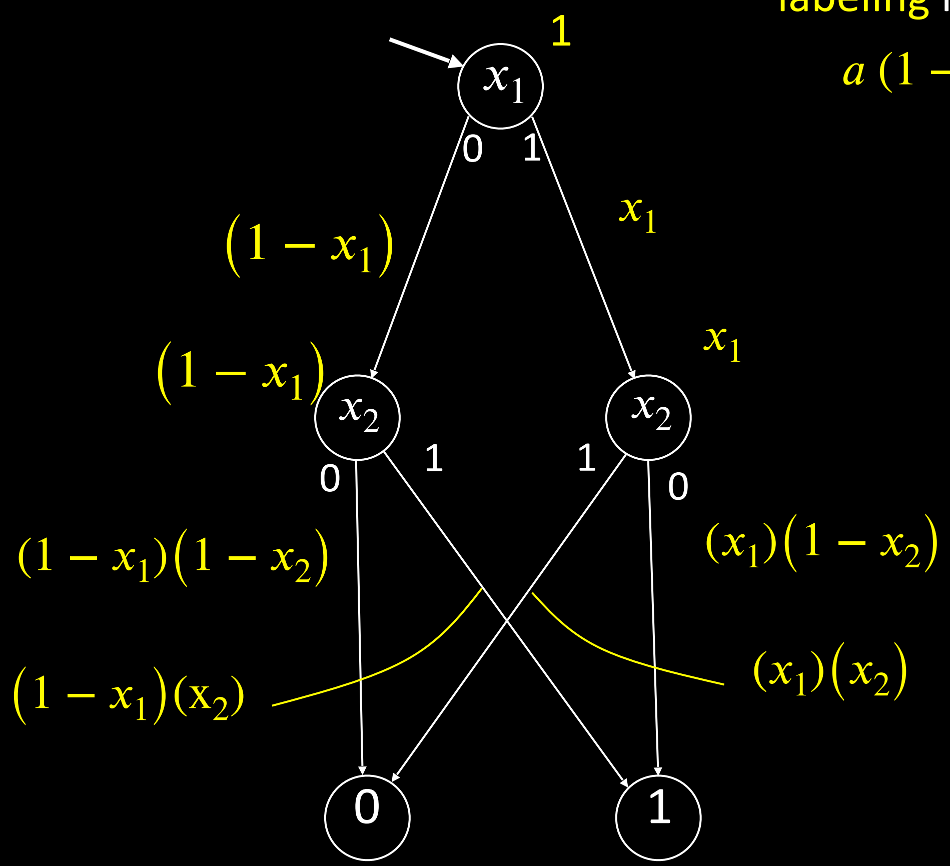


Recall labeling rules:

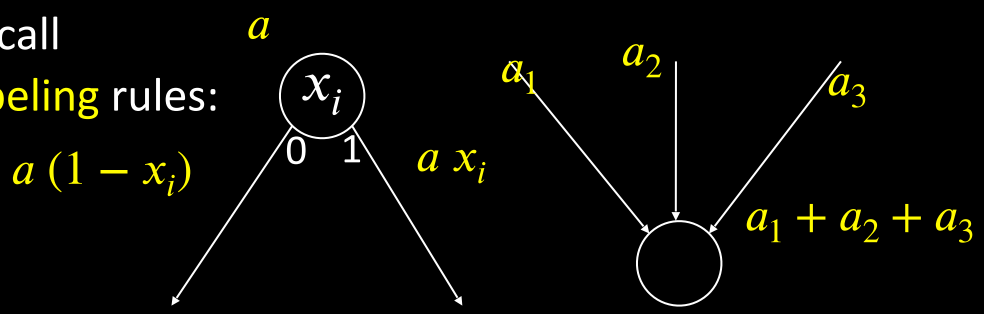


Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

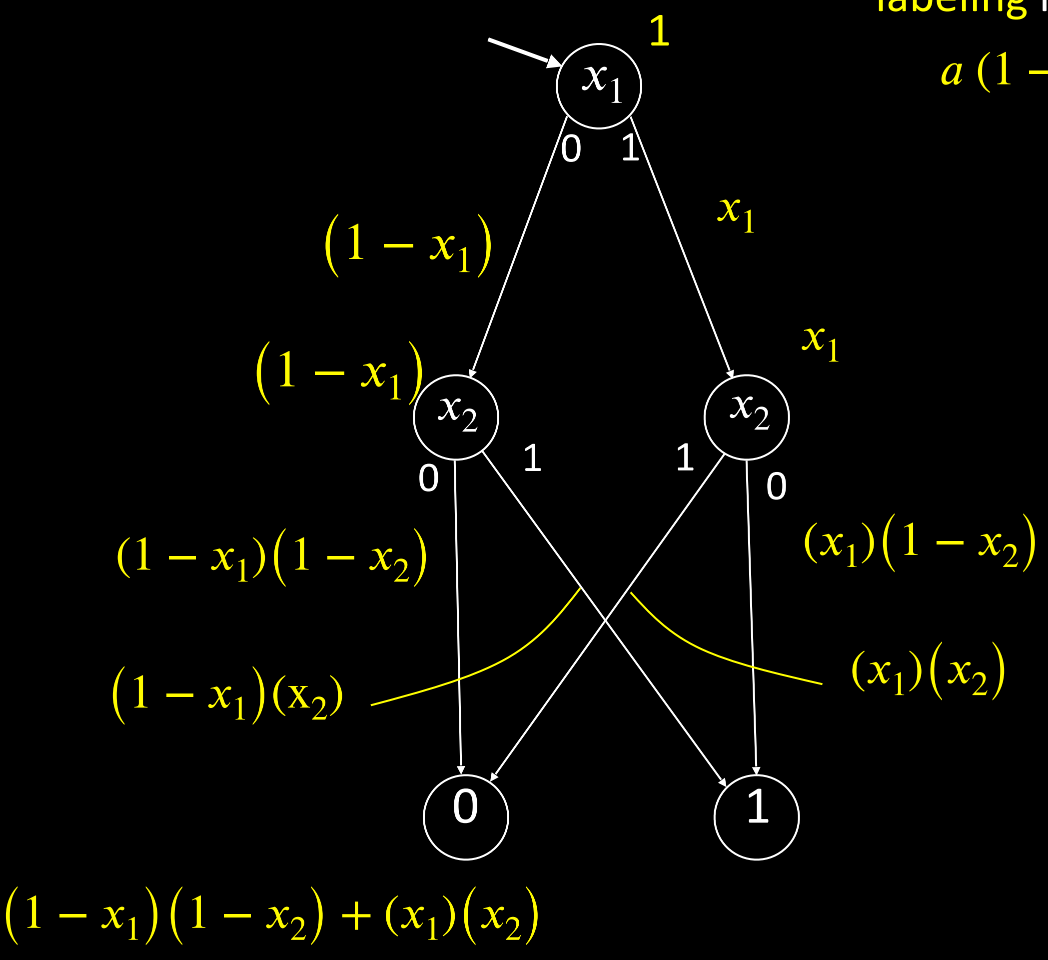


Recall labeling rules:



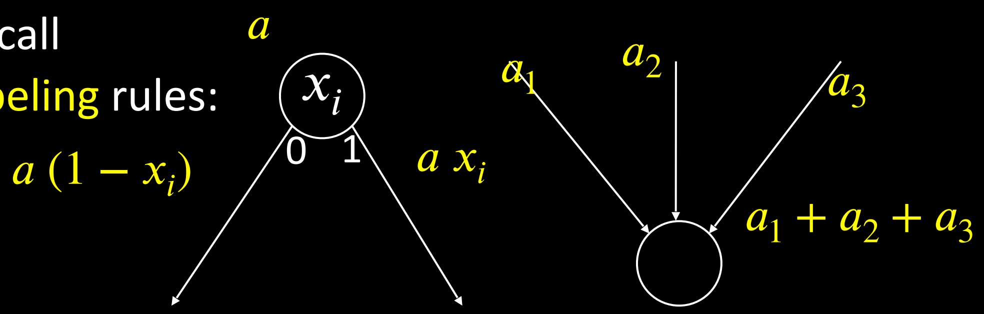
Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.



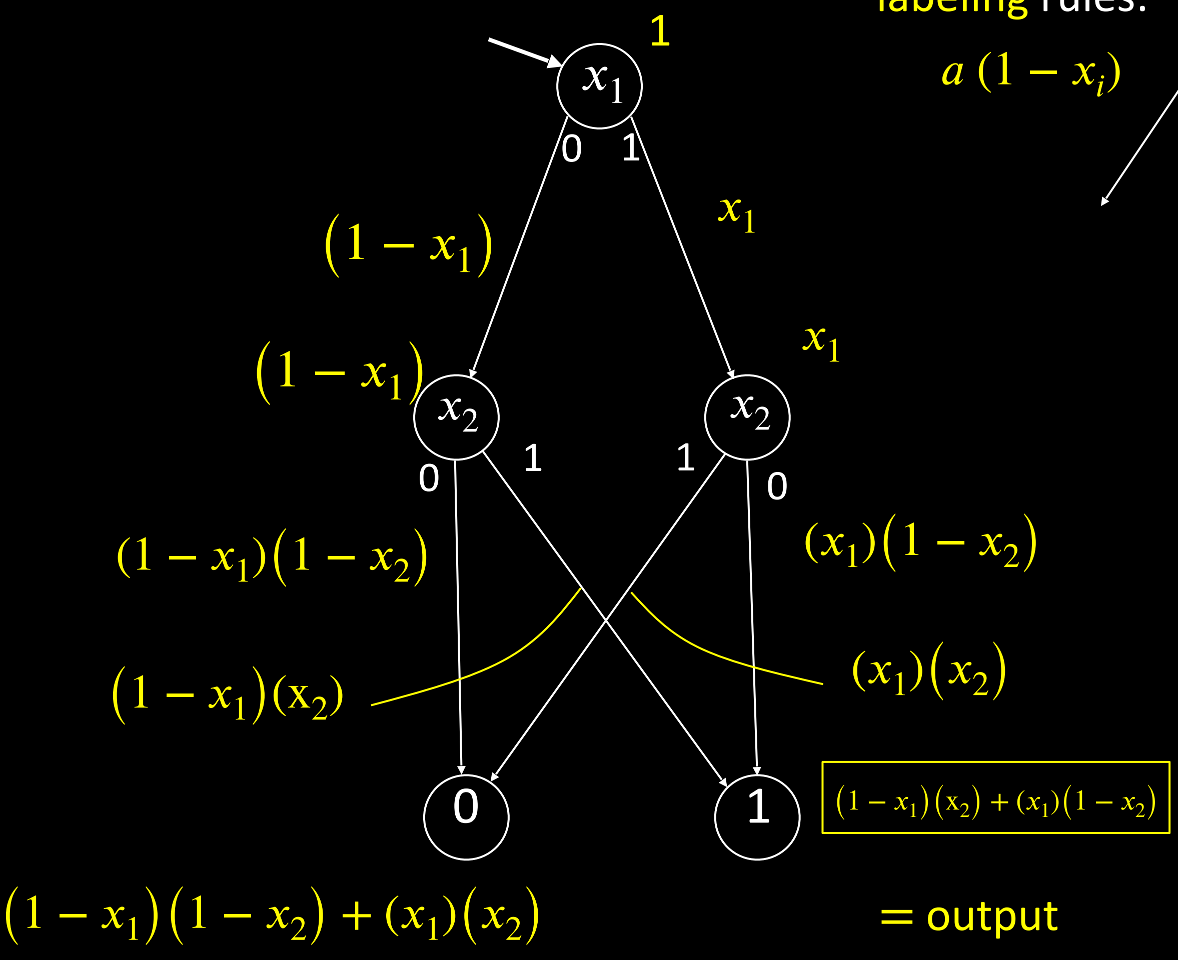
Recall

labeling rules:

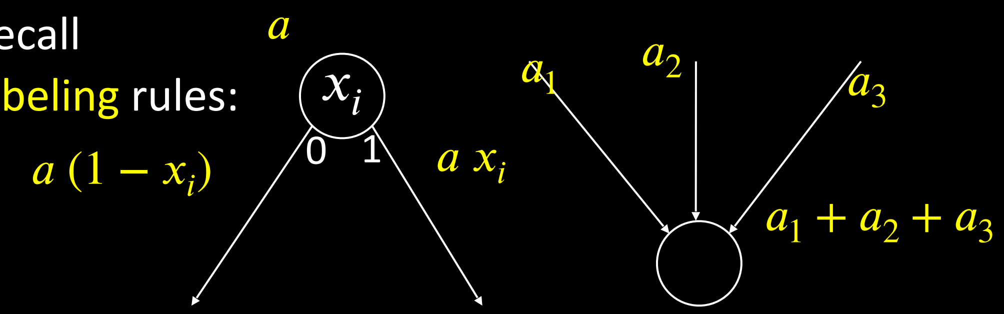


Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.



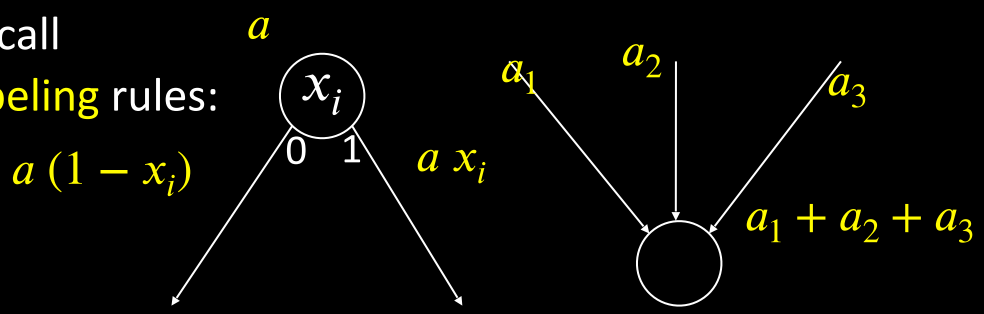
Recall labeling rules:

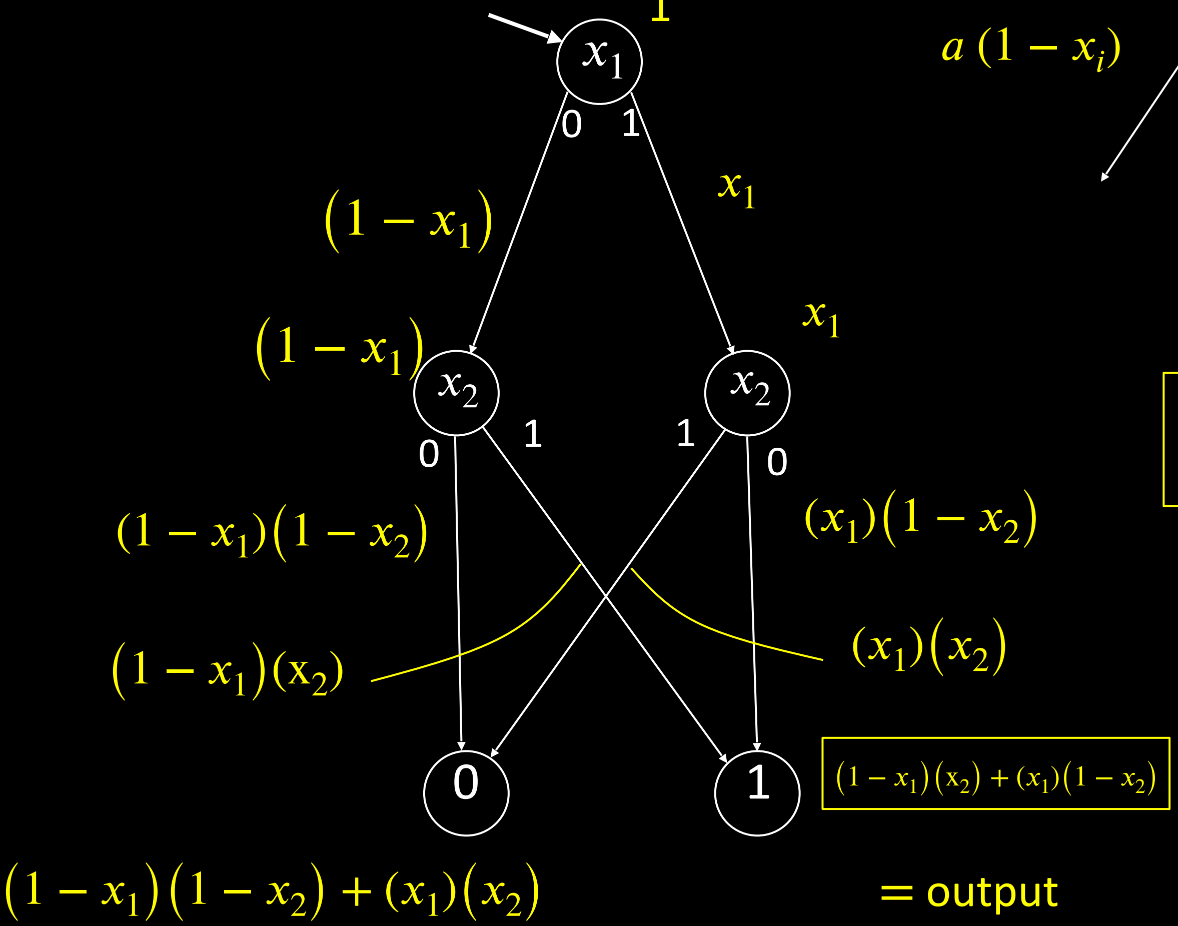


Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

Recall labeling rules:





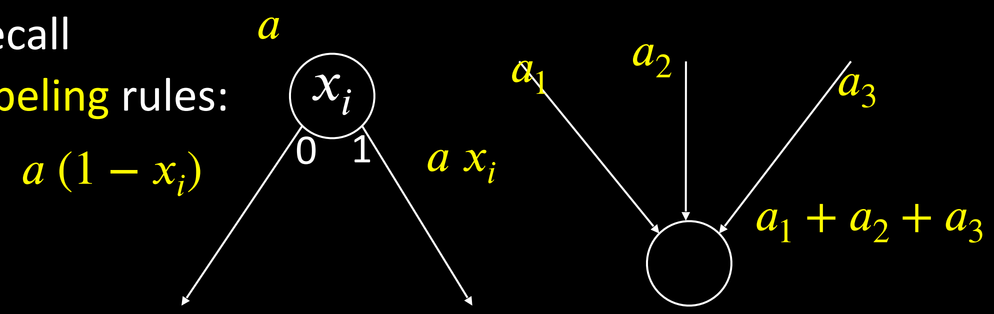
form of output

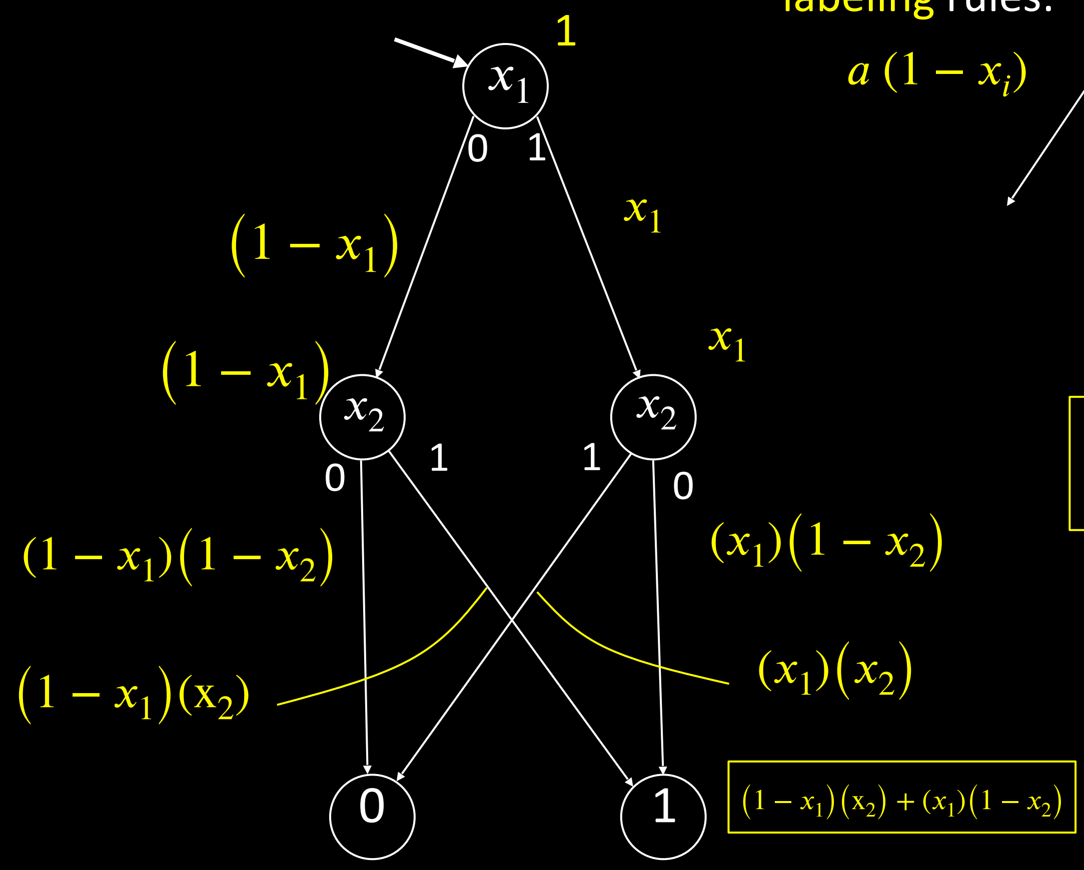
$$
\begin{aligned}
= \; & (1 - x_1)\;(x_2)\;(1 - x_3)\;(x_4)\;\cdots\;(1 - x_m) \\
+ \; & (x_1)\;(x_2)\;(x_3)\;(1 - x_4)\;\cdots\;(x_m) \\
+ \; & (x_1)\;(1 - x_2)(1 - x_3)\;(x_4)\;\cdots\;(x_m) \\
& \vdots \\
+ \; & (x_1)\;(x_2)\;(1 - x_3)\;(x_4)\;\cdots\;(x_m)
\end{aligned}
$$

Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

Recall labeling rules:





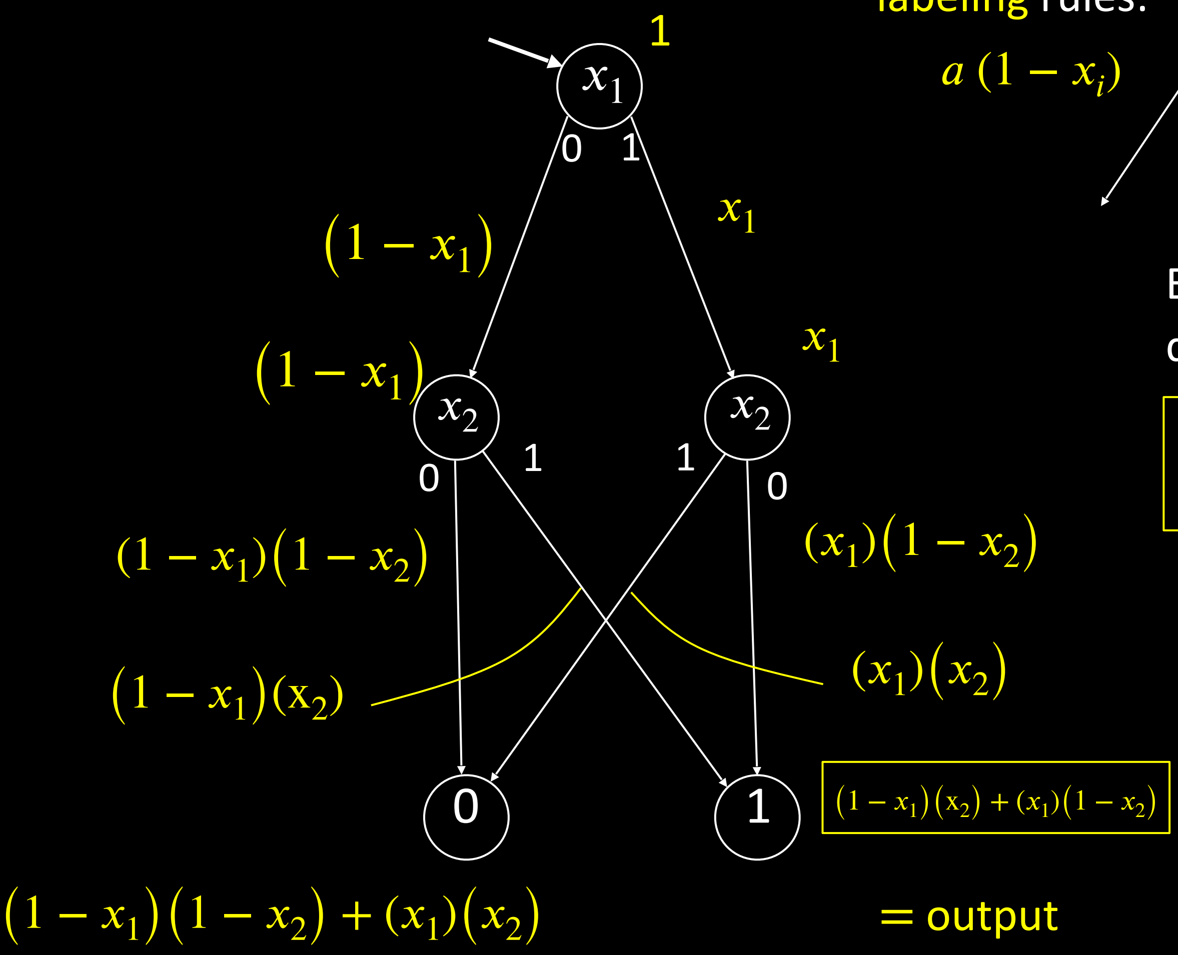
$(1-x_1)(1-x_2)+(x_1)(x_2)$ $=$ output

form of output

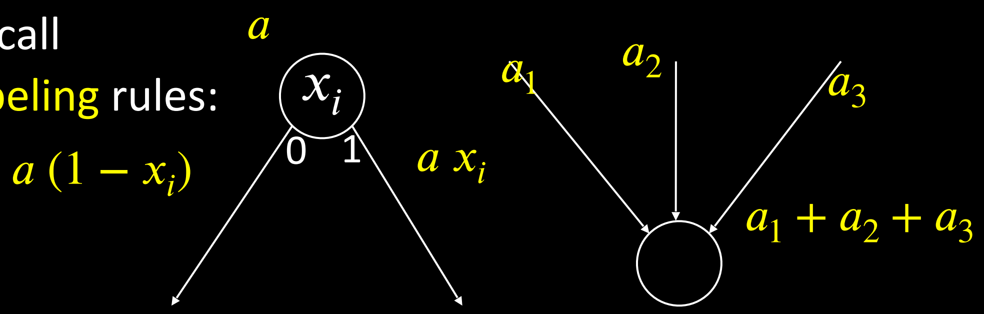
$$
\begin{aligned}
= \; & (1-x_1)\;(x_2)^3\;(1-x_3)\;(x_4)\;\cdots\;(1-x_m)\\
+ \; & (x_1)\;(x_2)\;(x_3)\;(1-x_4)\;\cdots\;(x_m)\\
+ \; & (x_1)\;(1-x_2)(1-x_3)\;(x_4)\;\cdots\;(x_m)\\
& \vdots\\
+ \; & (x_1)\;(x_2)\;(1-x_3)\;(x_4)\;\cdots\;(x_m)
\end{aligned}
$$

Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.



Recall labeling rules:



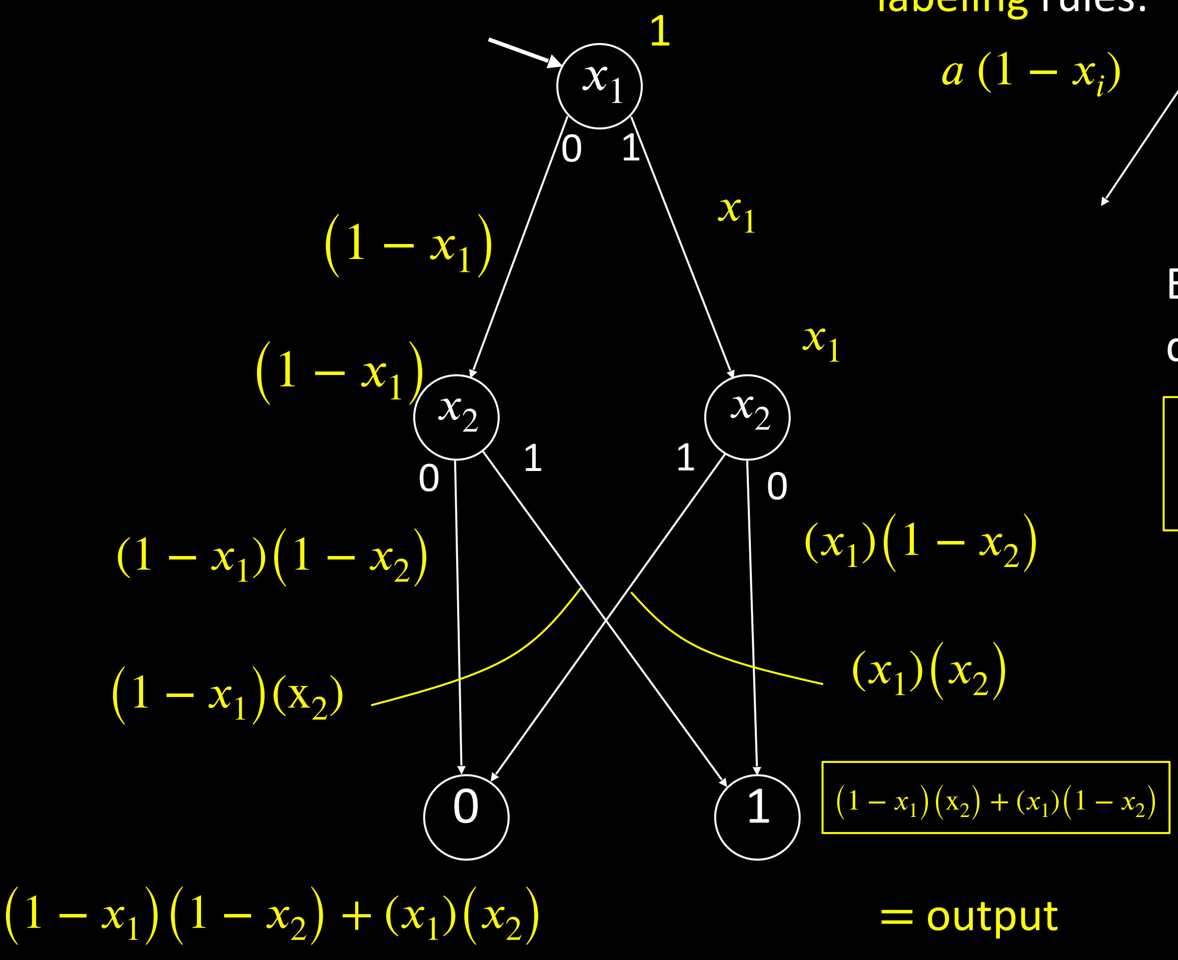
Exponents ≤ 1 due to "read-once"

$\boxed{\text{form of output}}$

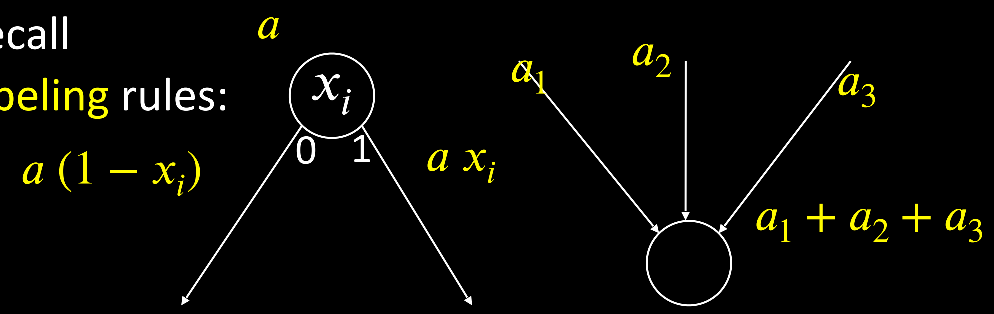
$$
\begin{aligned}
= \ & (1 - x_1)\ (x_2)^3\ (1 - x_3)\ (x_4)\ \cdots (1 - x_m) \\
+ \ & (x_1)\ (x_2)\ (x_3)\ (1 - x_4)\ \cdots\ (x_m) \\
+ \ & (x_1)\ (1 - x_2)(1 - x_3)\ (x_4)\ \cdots\ (x_m) \\
& \vdots \\
+ \ & (x_1)\ (x_2)\ (1 - x_3)\ (x_4)\ \cdots\ (x_m)
\end{aligned}
$$

Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.



Recall
labeling rules:



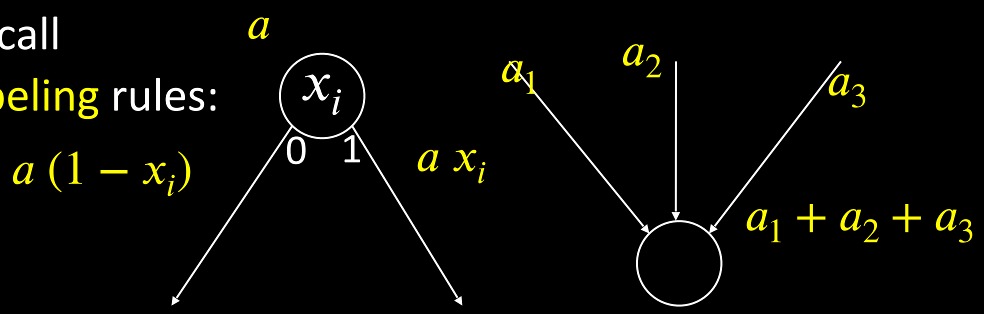
Exponents ≤ 1
due to "read-once"

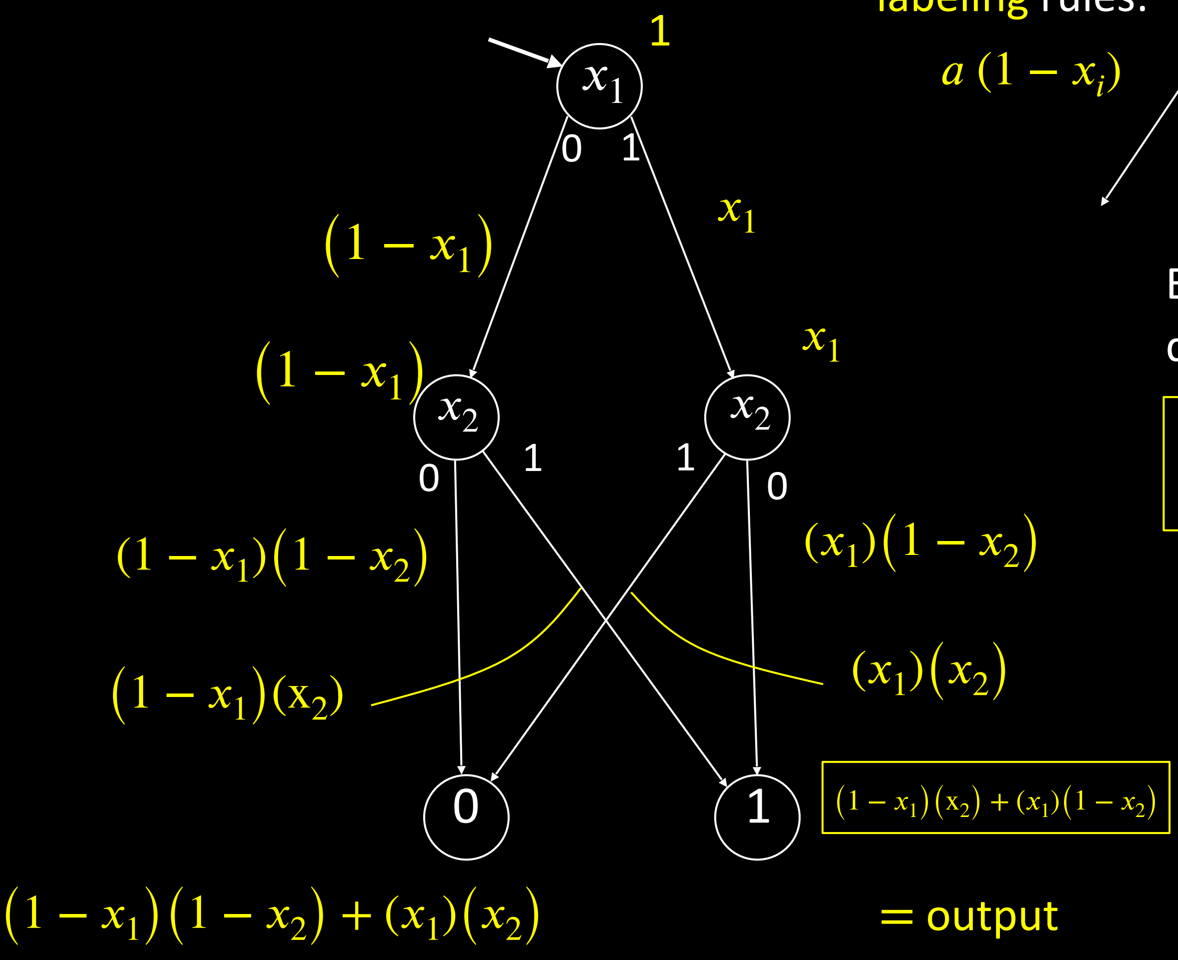
$$
\begin{aligned}
\boxed{\text{form of output}} = \; & (1-x_1)\,(x_2)^{\cancel{2}}\,(1-x_3)\,(x_4)\cdots(1-x_m) \\
+ \; & (x_1)\,(x_2)\,(x_3)\,(1-x_4)\cdots(x_m) \\
+ \; & (x_1)\,(1-x_2)(1-x_3)\,(x_4)\cdots(x_m) \\
& \vdots \\
+ \; & (x_1)\,(x_2)\,(1-x_3)\,(x_4)\cdots(x_m)
\end{aligned}
$$

Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

Recall labeling rules:





$=$ output

Exponents ≤ 1 due to "read-once"

Assume read <u>exactly</u> once so that for each i (x_i) or $(1-x_i)$ appears in every row

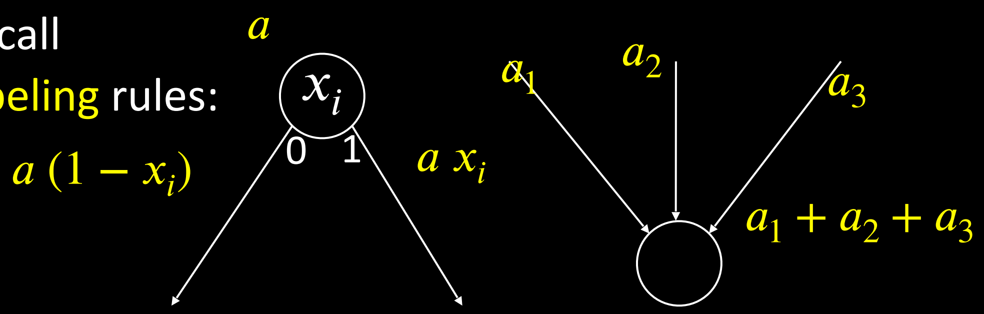
$\boxed{\text{form of output}}$

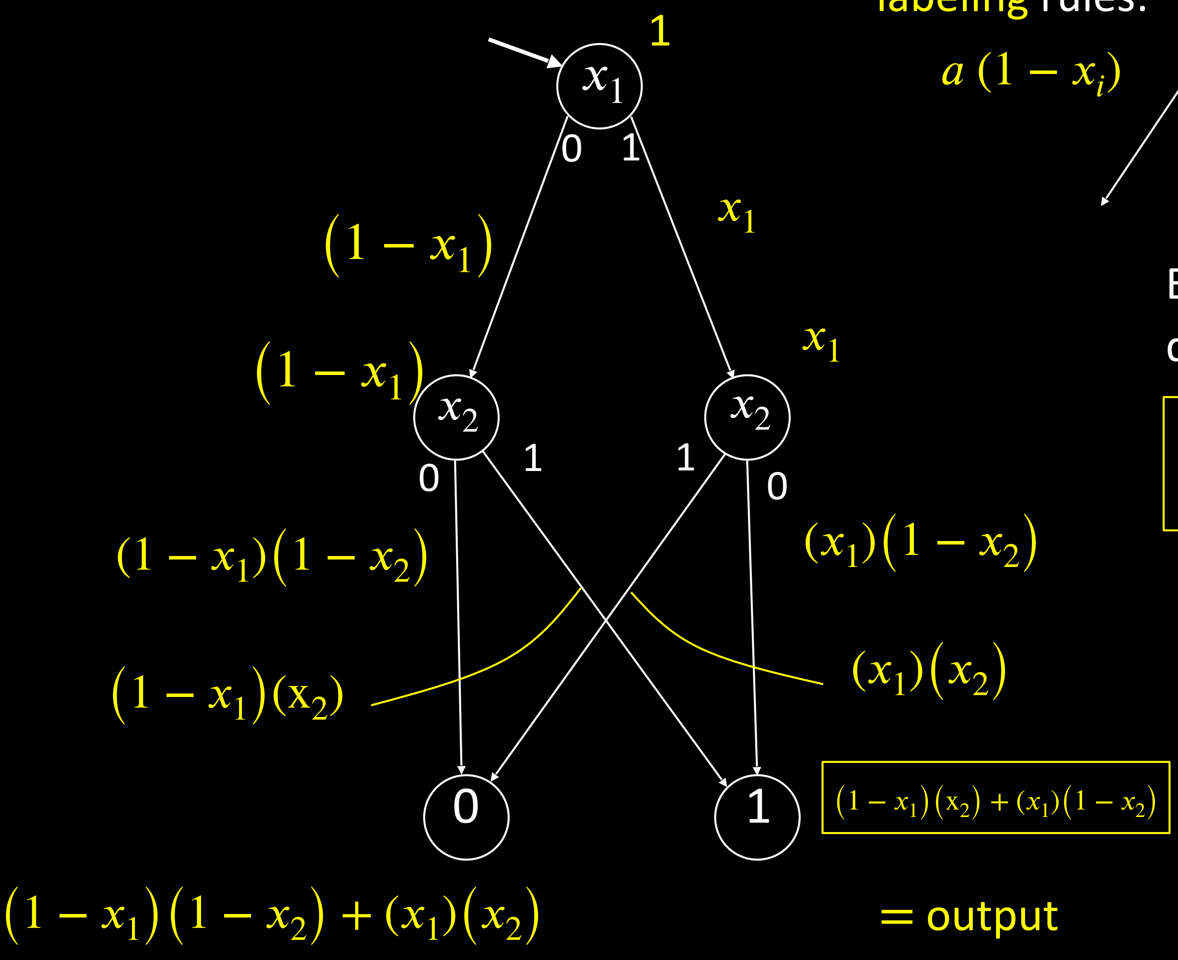
$$
\begin{aligned}
&= (1-x_1)\ (x_2)^{\cancel{2}}\ (1-x_3)\ (x_4)\ \cdots\ (1-x_m)\\
&+ (x_1)\ (x_2)\ (x_3)\ (1-x_4)\ \cdots\ (x_m)\\
&+ (x_1)\ (1-x_2)(1-x_3)\ (x_4)\ \cdots\ (x_m)\\
&\quad\vdots\\
&+ (x_1)\ (x_2)\ (1-x_3)\ (x_4)\ \cdots\ (x_m)
\end{aligned}
$$

Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

Recall **labeling rules:**





Exponents ≤ 1 due to "read-once"

Assume read <u>exactly</u> once so that for each i (x_i) or $(1-x_i)$ appears in every row

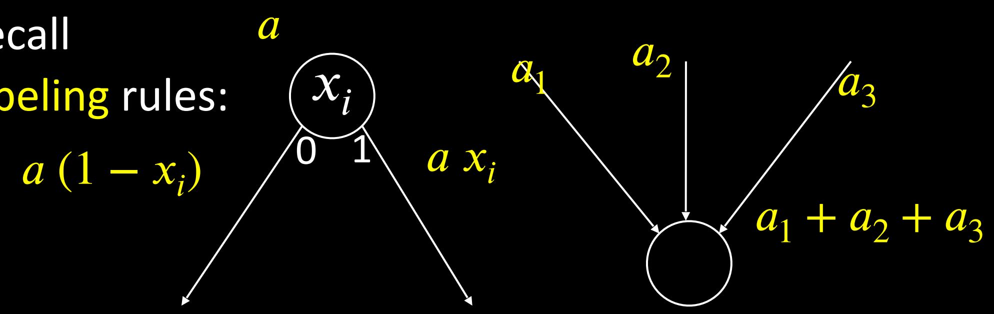
$$
\boxed{\text{form of output}}
\begin{aligned}
&= (1-x_1)\,(x_2)^{\cancel{2}}\,(1-x_3)\,(x_4)\,\cdots\,(1-x_m)\\
&+ (x_1)\,(x_2)\,(x_3)\,(1-x_4)\,\cdots\,(x_m)\\
&+ (x_1)\,(1-x_2)(1-x_3)\,(x_4)\,\cdots\,(x_m)\\
&\qquad\vdots\\
&+ (x_1)\,(x_2)\,(1-x_3)\,(x_4)\,\cdots\,(x_m)
\end{aligned}
$$

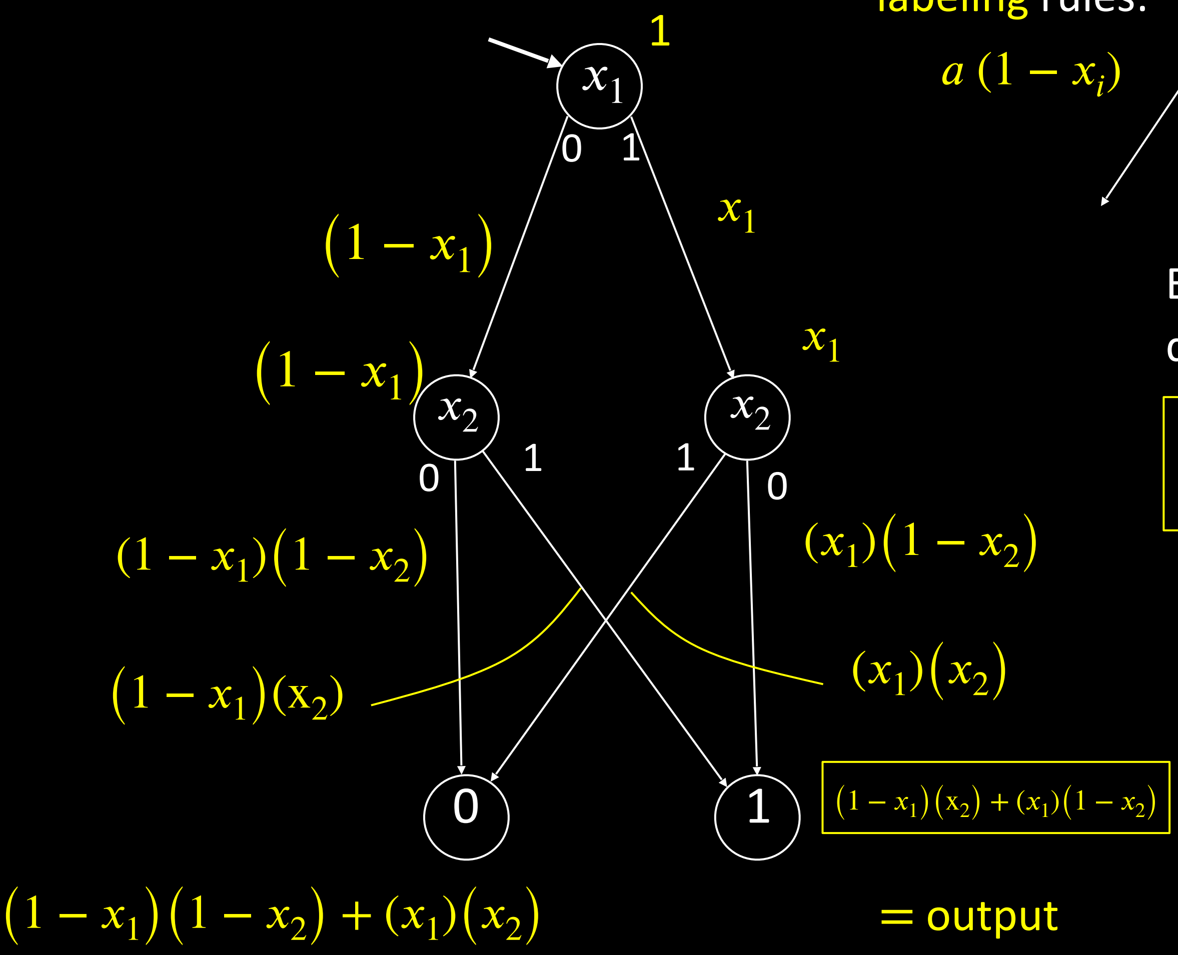
Corresponds to the TRUE rows in the truth table of the Boolean function

Symbolic Execution

Leave the x_i as variables and obtain an expression in the x_i for the output of the BP.

Recall labeling rules:





$= \text{output}$

Exponents ≤ 1 due to "read-once"

Assume read <u>exactly</u> once so that for each i (x_i) or $(1-x_i)$ appears in every row

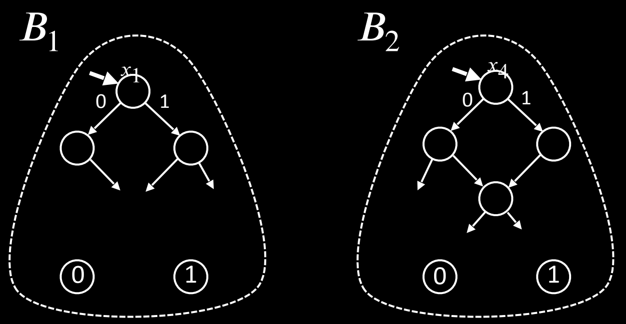
$$
\boxed{\text{form of output}}
\begin{aligned}
&= (1-x_1)\ (x_2)^{\cancel{2}}\ (1-x_3)\ (x_4)\ \cdots\ (1-x_m)\\
&+ (x_1)\ (x_2)\ (x_3)\ (1-x_4)\ \cdots\ (x_m)\\
&+ (x_1)\ (1-x_2)(1-x_3)\ (x_4)\ \cdots\ (x_m)\\
&\quad\vdots\\
&+ (x_1)\ (x_2)\ (1-x_3)\ (x_4)\ \cdots\ (x_m)
\end{aligned}
$$

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

EQROBP $\in$ BPP

Algorithm for EQROBP $=$ "On input $\langle B_1, B_2 \rangle$ [on variables $x_1, \ldots, x_m$]

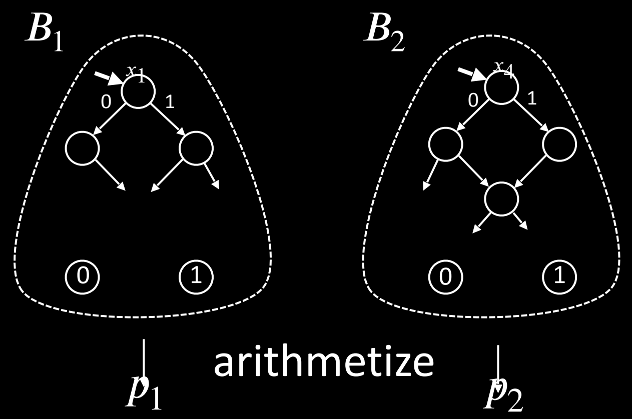
1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \ldots, r_m$ where each $r_i \in \mathbb{F}_q$.
3. Evaluate B_1 and B_2 on r by using arithmetization.
4. If B_1 and B_2 agree on r then *accept*.
   If they disagree then *reject*."



EQROBP $\in$ BPP

Algorithm for EQROBP $=$ "On input $\langle B_1, B_2\rangle$ [on variables $x_1, \ldots, x_m$]

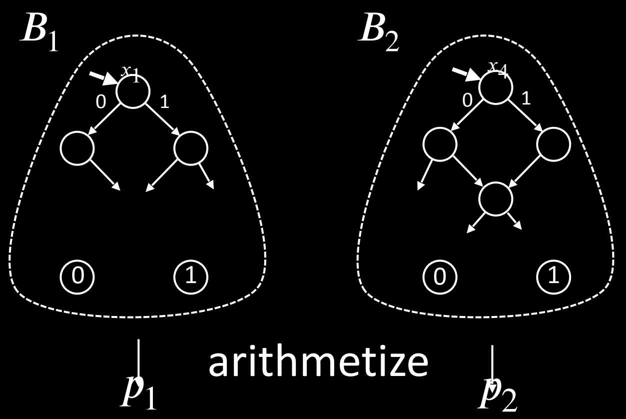
1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \ldots, r_m$ where each $r_i \in \mathbb{F}_q$.
3. Evaluate B_1 and B_2 on r by using arithmetization.
4. If B_1 and B_2 agree on r then *accept*.
   If they disagree then *reject*."



EQROBP $\in$ BPP

Algorithm for EQROBP $=$ "On input $\langle B_1, B_2 \rangle$ [on variables $x_1, \ldots, x_m$]

1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \ldots, r_m$ 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*."



p_1 and p_2 each have the form:

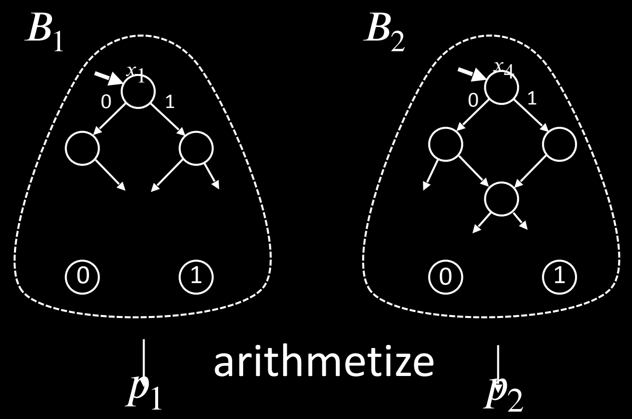
$$
\begin{aligned}
& (1-x_1)\ (x_2)\ (1-x_3)\ (x_4) \cdots (1-x_m) \\
+ \ & (x_1)\ (x_2)\ (x_3)\ (1-x_4) \cdots (x_m) \\
+ \ & (x_1)\ (1-x_2)(1-x_3)\ (x_4) \cdots (x_m) \\
& \vdots \\
+ \ & (x_1)\ (x_2)\ (1-x_3)\ (x_4) \cdots (x_m)
\end{aligned}
$$

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

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

1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \ldots, r_m$ 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) $B_1 \equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] = 1$



p_1 and p_2 each have the form:

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

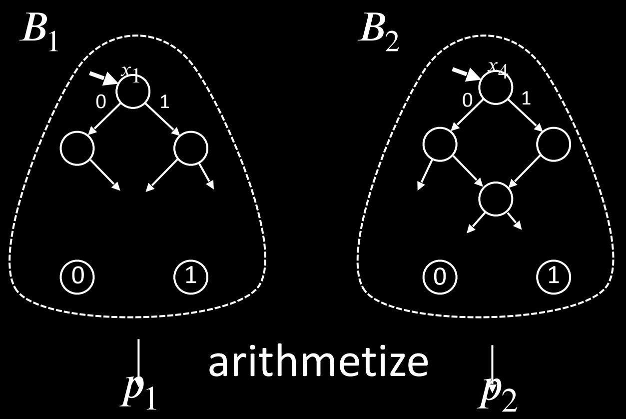
$EQ_{\text{ROBP}} \in$ BPP

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

1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \ldots, r_m$ 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) $B_1 \equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] = 1$

(2) $B_1 \not\equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] \leq \dfrac{1}{3}$



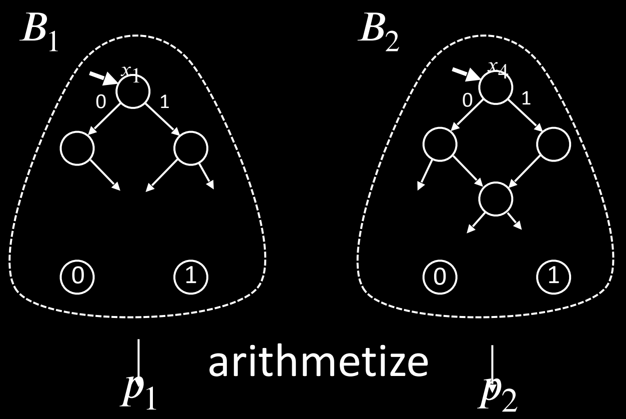
p_1 and p_2 each have the form:

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

EQROBP $\in$ BPP

Algorithm for EQROBP $=$ "On input $\langle B_1, B_2\rangle$ [on variables $x_1, \ldots, x_m$]

1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \ldots, r_m$ 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) $B_1 \equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] = 1$

(2) $B_1 \not\equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] \leq \frac{1}{3}$

Proof (1): If $B_1 \equiv B_2$ then they agree on all Boolean inputs.

p_1 and p_2 each have the form:

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

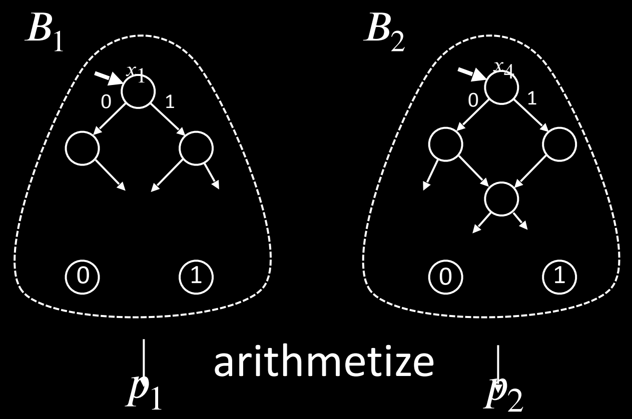
EQROBP $\in$ BPP

Algorithm for EQROBP $=$ "On input $\langle B_1, B_2 \rangle$ [on variables $x_1, \ldots, x_m$]

1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \ldots, r_m$ 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) $B_1 \equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] = 1$

(2) $B_1 \not\equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] \leq \dfrac{1}{3}$



Proof (1): If $B_1 \equiv B_2$ then they agree on all Boolean inputs.
Thus their functions have the same truth table.

p_1 and p_2 each have the form:

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

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

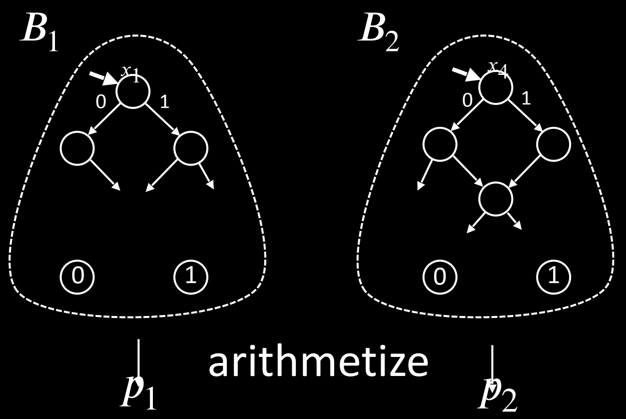
Algorithm for EQ_{ROBP} = "On input $\langle B_1, B_2 \rangle$ [on variables $x_1, \ldots, x_m$]

1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \ldots, r_m$ 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) $B_1 \equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] = 1$

(2) $B_1 \not\equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] \leq \frac{1}{3}$

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.



p_1 and p_2 each have the form:

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

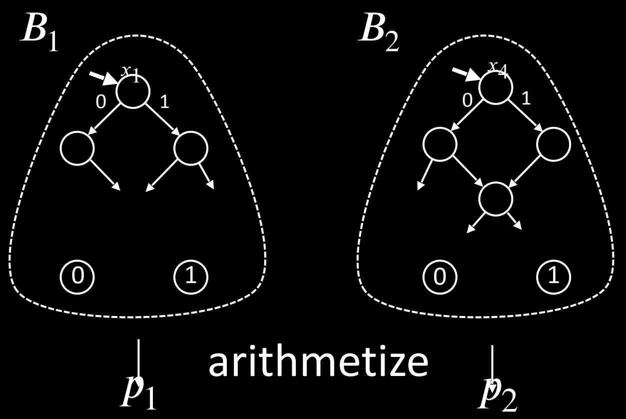
EQROBP $\in$ BPP

Algorithm for EQROBP $=$ "On input $\langle B_1, B_2 \rangle$ [on variables $x_1, \ldots, x_m$]

1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \ldots, r_m$ 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) $B_1 \equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] = 1$

(2) $B_1 \not\equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] \leq \frac{1}{3}$



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 <u>always</u> agree (even on non-Boolean inputs).

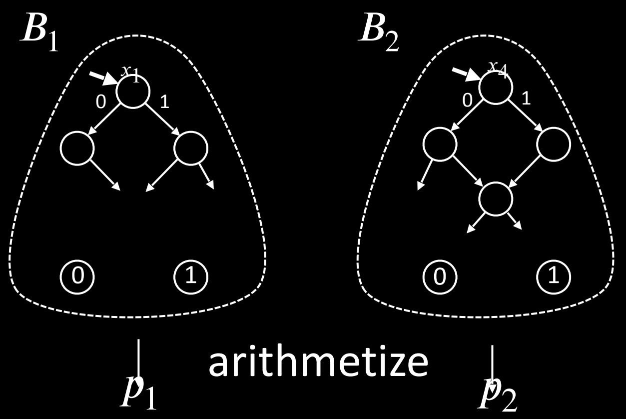
p_1 and p_2 each have the form:

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

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

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

1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \ldots, r_m$ 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) $B_1 \equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] = 1$

(2) $B_1 \not\equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] \leq \frac{1}{3}$

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 <u>always</u> agree (even on non-Boolean inputs).

Proof (2): If $B_1 \not\equiv B_2$ then $p_1 \neq p_2$ so $p = p_1 - p_2 \neq 0$.

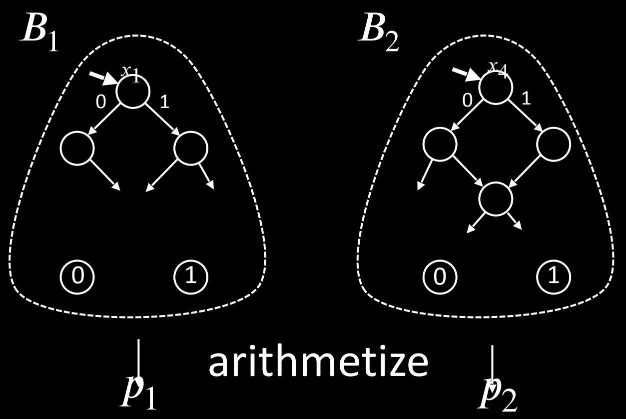
p_1 and p_2 each have the form:

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

EQROBP $\in$ BPP

Algorithm for EQROBP $=$ "On input $\langle B_1, B_2 \rangle$ [on variables $x_1, \ldots, x_m$]

1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \ldots, r_m$ 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) $B_1 \equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] = 1$

(2) $B_1 \not\equiv B_2 \rightarrow \Pr\left[\, p_1(r) = p_2(r) \,\right] \leq \dfrac{1}{3}$

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 <u>always</u> 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, $\Pr\left[\, p_1(r) = p_2(r) \,\right] \leq \dfrac{dm}{q} \leq \dfrac{m}{3m} = \dfrac{1}{3}$.

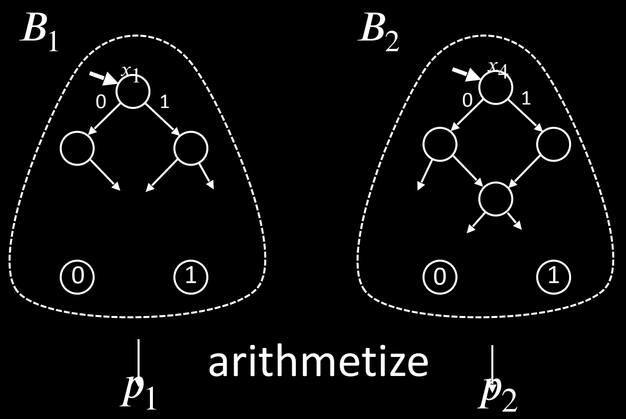
p_1 and p_2 each have the form:

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

EQROBP $\in$ BPP

Algorithm for EQROBP $=$ "On input $\langle B_1, B_2\rangle$ [on variables $x_1, \ldots, x_m$]

1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \ldots, r_m$ 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) $B_1 \equiv B_2 \to \Pr\left[\, p_1(r) = p_2(r) \,\right] = 1$

(2) $B_1 \not\equiv B_2 \to \Pr\left[\, p_1(r) = p_2(r) \,\right] \leq \dfrac{1}{3}$

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 <u>always</u> 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, $\Pr\left[\, p_1(r) = p_2(r) \,\right] \leq \dfrac{dm}{q} \leq \dfrac{m}{3m} = \dfrac{1}{3}$.

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

EQROBP $\in$ BPP

Algorithm for EQROBP $=$ "On input $\langle B_1, B_2 \rangle$ [on variables $x_1, \ldots, x_m$]

1. Find a prime $q \geq 3m$.
2. Pick a random *non-Boolean* input assignment $r = r_1, \ldots, r_m$ 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) $B_1 \equiv B_2 \rightarrow$ $\Pr\left[\, p_1(r) = p_2(r) \,\right] = 1$

(2) $B_1 \not\equiv B_2 \rightarrow$ $\Pr\left[\, p_1(r) = p_2(r) \,\right] \leq \frac{1}{3}$

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

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 <u>always</u> agree (even on non-Boolean inputs).

Proof (2): If $B_1 \not\equiv B_2$ then $p_1 \neq p_2$ so $p = p_1 - p_2 \neq 0$.

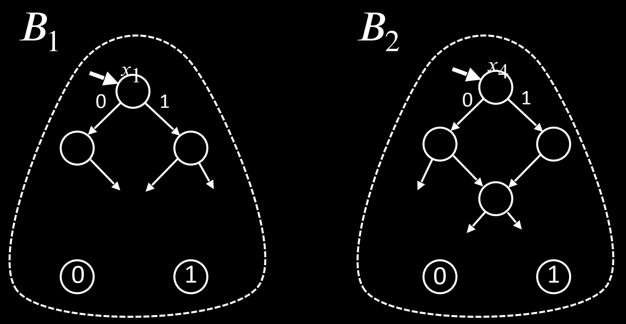
From Schwartz-Zippel, $\Pr\left[\, p_1(r) = p_2(r) \,\right] \leq \frac{dm}{q} \leq \frac{m}{3m} = \frac{1}{3}$.

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



p_1 arithmetize p_2

Check-in 24.3

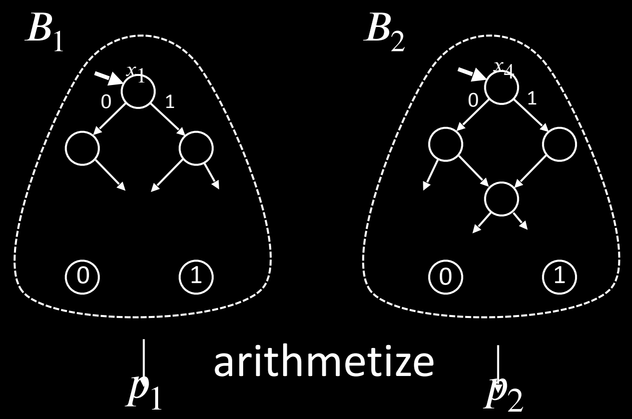
$EQ_{\text{ROBP}} \in$ BPP

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.



Quick review of today

1. Simulated Read-once Branching Programs by polynomials
2. Gave probabilistic polynomial equality testing method
3. Showed EQROBP $\in$ BPP

Quick review of today

1. Simulated Read-once Branching Programs by polynomials
2. Gave probabilistic polynomial equality testing method
3. Showed EQROBP $\in$ BPP