

SAT

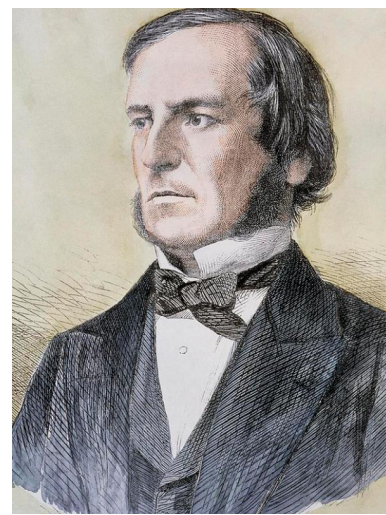
a different kind of search problem

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

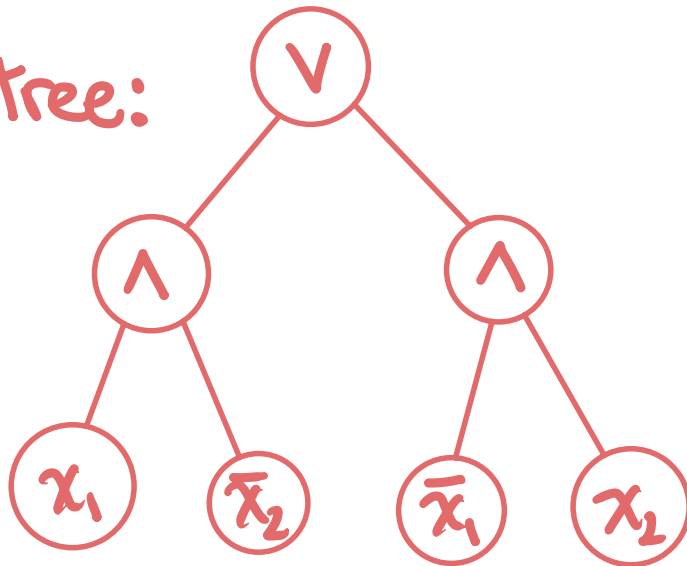


George Boole,
1815-1864

XOR ("exclusive-or") (TRUE iff $x_1 \neq x_2$)

$$f(x_1, x_2) = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$$

as a tree:



Adding two bits

$$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if } x_1 + x_2 = x_3 x_4 \\ 0 & \text{otherwise} \end{cases}$$

$$f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2 \wedge x_3 \wedge \bar{x}_4) \vee (\neg(x_1 \wedge x_2) \wedge ((x_1 \vee x_2) \wedge x_4) \vee (\neg(x_1 \vee x_2) \wedge \bar{x}_4))$$

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

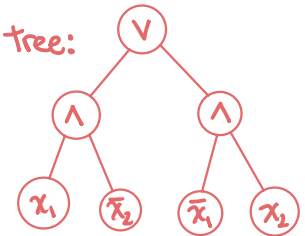
"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

XOR ("exclusive-or") (TRUE iff $x_1 \neq x_2$)

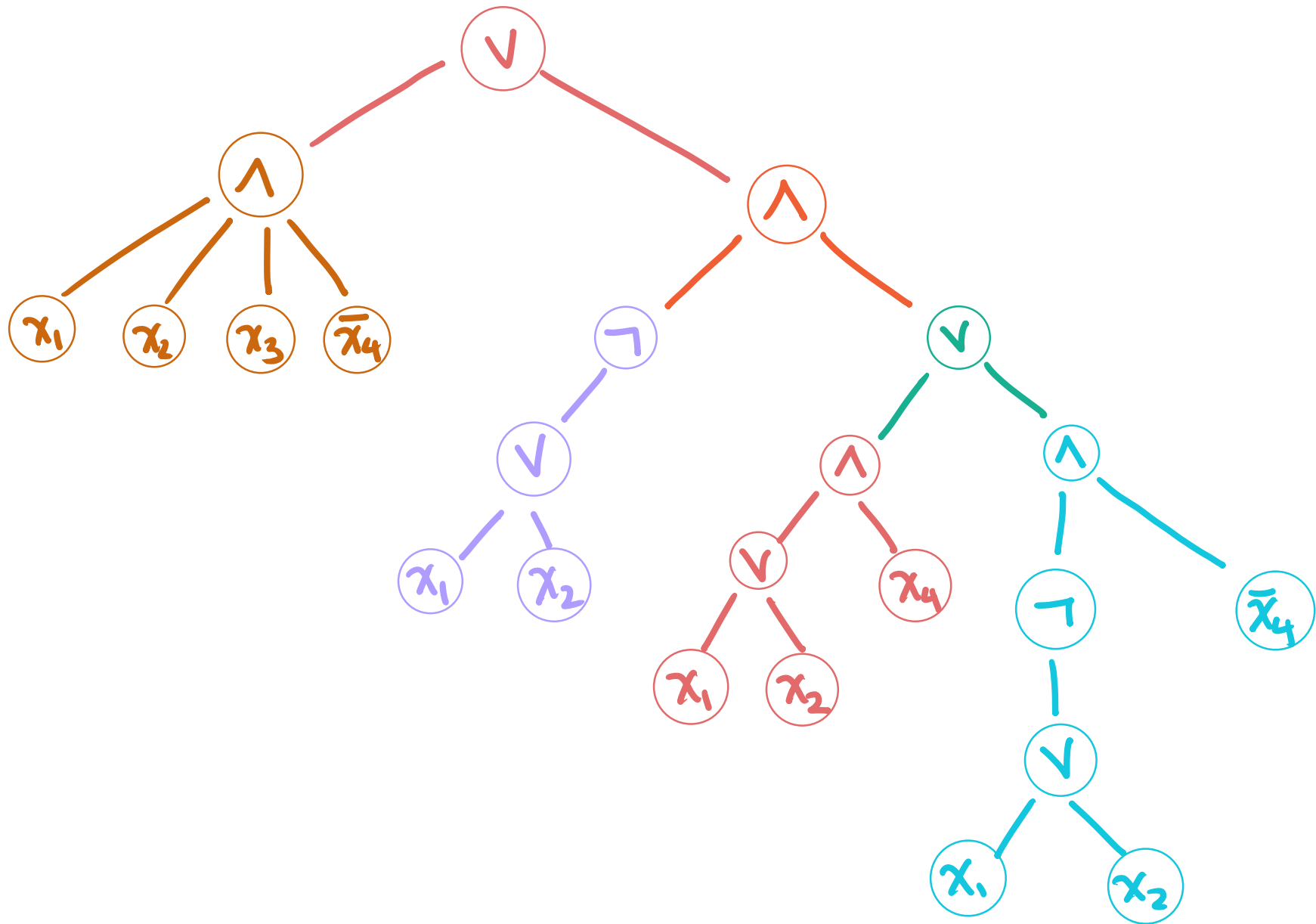
$$f(x_1, x_2) = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$$

as a tree:



Adding two bits $f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if } x_1 + x_2 = x_3 x_4 \\ 0 & \text{otherwise} \end{cases}$

$$f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2 \wedge x_3 \wedge \bar{x}_4) \vee (\neg(x_1 \wedge x_2) \wedge ((x_1 \vee x_2) \wedge x_4) \vee (\neg(x_1 \vee x_2) \wedge \bar{x}_4))$$



"SAT Programming Language"

"x and y" $x \wedge y$
 "x or y" $x \vee y$
 "not x" $\bar{x}$ (or $\neg x$)

★ "z = x" "both true or both false"
 $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$

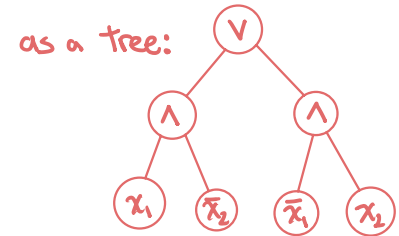
★ "if x then y else z" "(x=1, y=1) or (x=0, z=1)"
 $(x \wedge y) \vee (\bar{x} \wedge z)$

So Boolean algebra is very expressive

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$
 "x or y" $x \vee y$
 "not x" $\bar{x}$ (or $\neg x$)

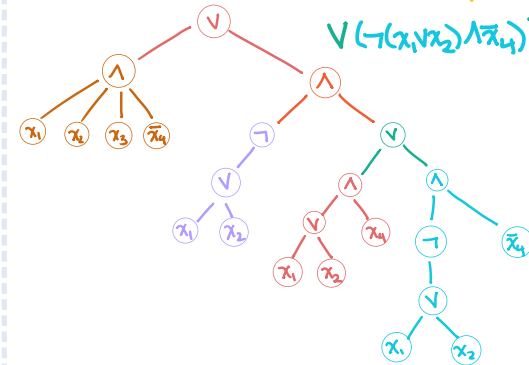
XOR ("exclusive-or") (TRUE iff $x_1 \neq x_2$)
 $f(x_1, x_2) = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$



Adding two bits

$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if } x_1 + x_2 = x_3 x_4 \\ 0 & \text{otherwise} \end{cases}$

$f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2 \wedge x_3 \wedge \bar{x}_4) \vee (\neg(x_1 \wedge x_2) \wedge ((x_1 \vee x_2) \wedge x_4) \vee (\neg(x_1 \vee x_2) \wedge \bar{x}_4))$



Boolean Satisfiability

(informally)

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

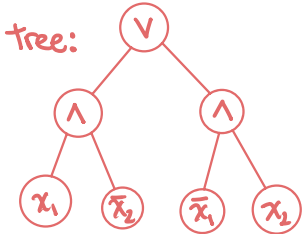
"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

XOR ("exclusive-or") (TRUE iff $x_1 \neq x_2$)

$$f(x_1, x_2) = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$$

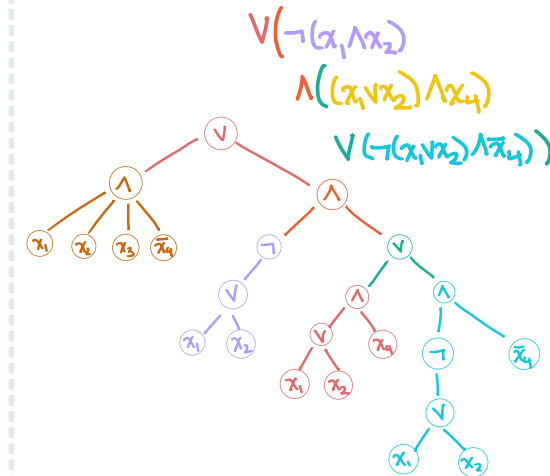
as a tree:



Adding two bits

$$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if } x_1 + x_2 = x_3 x_4 \\ 0 & \text{otherwise} \end{cases}$$

$$f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2 \wedge x_3 \wedge \bar{x}_4) \vee (\neg(x_1 \wedge x_2) \wedge ((x_1 \vee x_2) \wedge x_4)) \vee (\neg(x_1 \vee x_2) \wedge \bar{x}_4)$$



"SAT Programming Language"

★ "z = x"

"both true or both false"

$$(z \wedge x) \vee (\bar{z} \wedge \bar{x})$$

★ "if x then y else z"

"(x=1, y=1) or (x=0, z=1)"

$$(x \wedge y) \vee (\bar{x} \wedge z)$$

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$

• n variables • "size" m

e.g. $\varphi(\overset{\text{(false)}}{0}, \overset{\text{(true)}}{1}, 0, 0, \neg, 1, 1) = 0$ (false)

Goal: decide if φ is "satisfiable";
i.e., if there exists $x_1, x_2, \dots, x_n \in \{0, 1\}$
s.t. $\varphi(x_1, x_2, \dots, x_n) = 1$

e.g. $\varphi(x_1, x_2) =$

$$(x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2)$$

Boolean Satisfiability

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$

- n variables
- "size" m

e.g. $\varphi(\overset{\text{(false)}}{0}, \overset{\text{(true)}}{1}, 0, 0, \neg, \overset{\text{(false)}}{1}, 1) = 0$

Goal: decide if φ is "satisfiable";
i.e., if there exists $x_1, x_2, \dots, x_n \in \{0, 1\}$
s.t. $\varphi(x_1, x_2, \dots, x_n) = 1$

"search problem": we can recognize an answer (e.g. satisfying assignment x) when we see it (by evaluating $f(x)$).

- at the very least: we can try all x .
- harder "search problem" than "over/under"

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

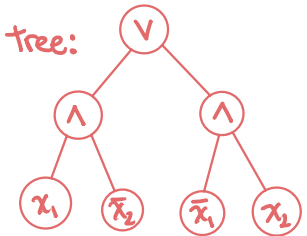
"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

XOR ("exclusive-or") (TRUE iff $x_1 \neq x_2$)

$$f(x_1, x_2) = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$$

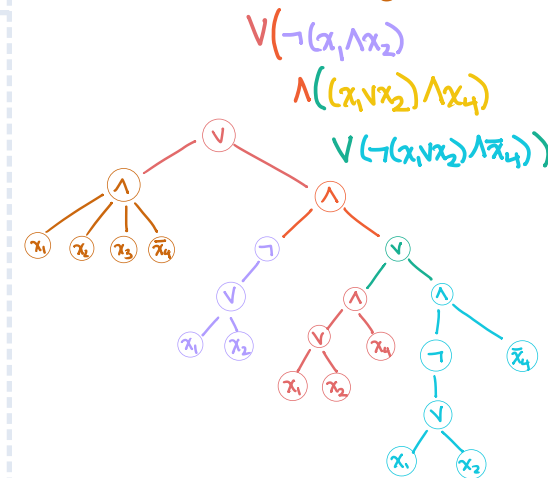
as a tree:



Adding two bits

$$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if } x_1 + x_2 = x_3 + x_4 \\ 0 & \text{otherwise} \end{cases}$$

$$f(x_1, x_2, x_3, x_4) = (x_1 \wedge x_2 \wedge x_3 \wedge \bar{x}_4) \vee (\neg(x_1 \wedge x_2) \wedge ((x_1 \vee x_2) \wedge x_4) \vee (\neg(x_1 \vee x_2) \wedge \bar{x}_4))$$



"SAT Programming Language"

★ "z = x"

"both true or both false"

$$(z \wedge x) \vee (\bar{z} \wedge \bar{x})$$

★ "if x then y else z"

"(x=1, y=1) or (x=0, z=1)"

$$(x \wedge y) \vee (\bar{x} \wedge z)$$

Boolean Satisfiability

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$

- n variables
- "size" m

e.g. $\varphi(\overset{\text{(false)}}{0}, \overset{\text{(true)}}{1}, 0, 0, \neg, \overset{\text{(false)}}{1}, 1) = 0$

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

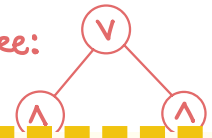
"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

XOR ("exclusive-or") (TRUE iff $x_1 \neq x_2$)

$$f(x_1, x_2) = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$$

as a tree:



Goal: The Big Question

Is there a polynomial time algorithm for Boolean satisfiability?

x_2

$x_2 = x_3 x_4$

wise

$x_3 \wedge \bar{x}_4$

x_2

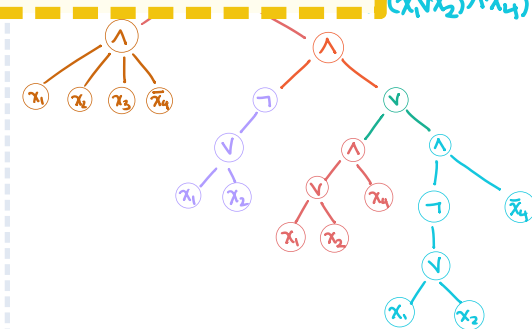
$x_2 \wedge x_4$

$(x_1 \vee x_2) \wedge \bar{x}_4$

answer (e.g. satisfying assignment x) when we see it (by evaluating $f(x)$).

- at the very least: we can try all x .

- harder "search problem" than "over/under"



"SAT Programming Language"

★ "z = x"

"both true or both false"
 $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$

★ "if x then y else z"

"(x=1, y=1) or (x=0, z=1)"
 $(x \wedge y) \vee (\bar{x} \wedge z)$

The Big Question

Is there a polynomial time algorithm for Boolean satisfiability?

Ideas?

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

"SAT Programming Language"

★ "z = x" $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$ "both true or both false"

★ "if x then y else z" $(x \wedge y) \vee (\bar{x} \wedge z)$ "(x=1, y=1) or (x=0, z=1)"

Boolean Satisfiability

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$
• n variables • "size" m

e.g. $\varphi(0, 1, 0, 0, \dots, 1, 1) = 0$ (false) (true) (false)

Goal: decide if φ is "satisfiable";
i.e., if there exists $x_1, x_2, \dots, x_n \in \{0, 1\}$
s.t. $\varphi(x_1, x_2, \dots, x_n) = 1$

"search problem": we can recognize an answer (e.g. satisfying assignment x) when we see it (by evaluating $\varphi(x)$).
• at the very least: we can try all x .
• harder "search problem" than "over/under"

The Big Question

Is there a polynomial time algorithm for Boolean satisfiability?

Pros

Boolean logic is very simple, natural for computing

Easy to evaluate boolean functions

Cons

Boolean logic is very expressive, universal

too good to be true?

OK, what about special cases?

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

"SAT Programming Language"

★ "z = x" $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$ "both true or both false"

★ "if x then y else z" $(x \wedge y) \vee (\bar{x} \wedge z)$ "(x=1, y=1) or (x=0, z=1)"

Boolean Satisfiability

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$
• n variables • "size" m

e.g. $\varphi(0, 1, 0, 0, \dots, 1, 1) = 0$ (false) (true) (false)

Goal: decide if φ is "satisfiable";
i.e., if there exists $x_1, x_2, \dots, x_n \in \{0, 1\}$
s.t. $\varphi(x_1, x_2, \dots, x_n) = 1$

"search problem": we can recognize an answer (e.g. satisfying assignment x) when we see it (by evaluating $\varphi(x)$).

• at the very least: we can try all x .
• harder "search problem" than "over/under"

The Big Question

Is there a polynomial time algorithm
for Boolean satisfiability?

The Big Question

Is there a polynomial time algorithm
for Boolean satisfiability?

CNF (Conjunctive Normal Form)

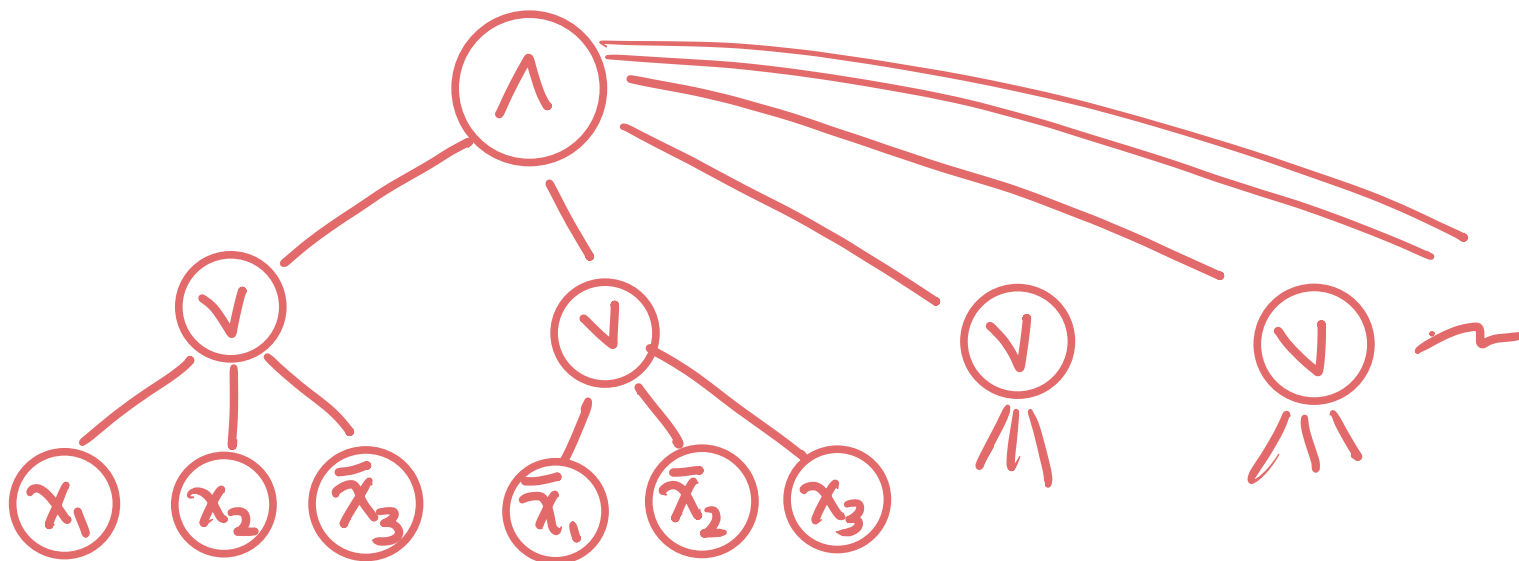
"and - or's"

$$f(x_1, \dots, x_n) =$$

"clause"

$$(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\neg \vee \neg) \wedge \dots$$

CNF's have "flat trees"



Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

"SAT Programming Language"

★ "z=x" $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$
"both true or both false"

★ "if x then y else z" $(x \wedge y) \vee (\bar{x} \wedge z)$
"(x=1, y=1) or (x=0, z=1)"

Boolean Satisfiability

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$
• n variables • "size" m

e.g. $\varphi(0, 1, 0, 0, \dots, 1, 1) = 0$
(false) (true) (false)

Goal: decide if φ is "satisfiable";
i.e., if there exists $x_1, x_2, \dots, x_n \in \{0, 1\}$
s.t. $\varphi(x_1, x_2, \dots, x_n) = 1$

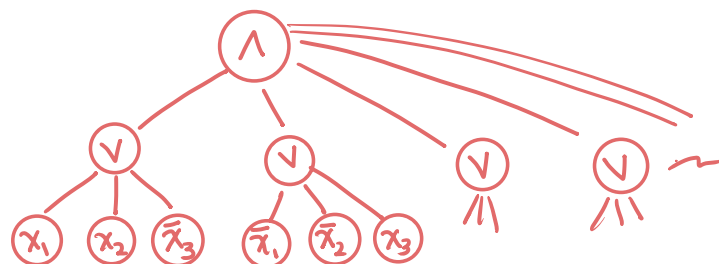
"search problem": we can recognize an answer (e.g. satisfying assignment x) when we see it (by evaluating $\varphi(x)$).
• at the very least: we can try all x .
• harder "search problem" than "over/under"

The Big Question

Is there a polynomial time algorithm for Boolean satisfiability?

CNF (Conjunctive Normal Form)

"and-of-or's"



"clause"

$$f(x_1, \dots, x_n) =$$

$$(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\neg \vee \neg) \wedge \neg$$

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

"SAT Programming Language"

★ "z=x" $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$

★ "if x then y else z" $(x \wedge y) \vee (\bar{x} \wedge z)$

Boolean Satisfiability

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$
• n variables • "size" m

e.g. $\varphi(0, 1, 0, 0, \neg, 1, 1) = 0$

Goal: decide if φ is "satisfiable";
i.e., if there exists $x_1, x_2, \dots, x_n \in \{0, 1\}$
s.t. $\varphi(x_1, x_2, \dots, x_n) = 1$

"search problem": we can recognize an answer (e.g. satisfying assignment x) when we see it (by evaluating $\varphi(x)$).
• at the very least: we can try all x .
• harder "search problem" than "over/under"

The Big Question

Is there a polynomial time algorithm for Boolean satisfiability?

(CNF $\rightarrow$) SAT problem

Given boolean formula f in CNF, decide if f is satisfiable

More modest Question

Is there a polynomial time algorithm for CNF-satisfiability?

"CNF-SAT Programming Language"

"x and y"

$$x \wedge y$$

"x or y"

$$x \vee y$$

"not x"

$$\bar{x} \quad (\text{or } \neg x)$$

★ "z = x"

"both true or both false"

$$(z \wedge x) \vee (\bar{z} \wedge \bar{x})$$

$$= (z \vee (\bar{z} \wedge \bar{x})) \wedge (x \vee (\bar{z} \wedge \bar{x}))$$

$$= (z \vee \bar{z}) \wedge (z \vee \bar{x}) \wedge (x \vee \bar{z}) \wedge (x \vee \bar{x})$$

$$= (z \vee \bar{x}) \wedge (x \vee \bar{z})$$

★ "if x then y else z"

"(x=1, y=1) or (x=0, z=1)"

$$(x \wedge y) \vee (\bar{x} \wedge z)$$

$$= (x \vee \bar{x}) \wedge (y \vee \bar{x}) \wedge (x \vee z) \wedge (x \vee \bar{z})$$

$$= (y \vee \bar{x}) \wedge (x \vee z) \wedge (x \vee \bar{z})$$

So CNF is also very expressive

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

"SAT Programming Language"

★ "z = x" $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$
"both true or both false"

★ "if x then y else z" $(x \wedge y) \vee (\bar{x} \wedge z)$
"(x=1, y=1) or (x=0, z=1)"

Boolean Satisfiability

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$
 • n variables • "size" m

e.g. $\varphi(0, 1, 0, 0, \dots, 1, 1) = 0$
(false) (true) (false)

Goal: decide if φ is "satisfiable";
 i.e., if there exists $x_1, x_2, \dots, x_n \in \{0, 1\}$
 s.t. $\varphi(x_1, x_2, \dots, x_n) = 1$

"search problem": we can recognize an answer (e.g. satisfying assignment x) when we see it (by evaluating $\varphi(x)$).
 • at the very least: we can try all x .
 • harder "search problem" than "over/under"

The Big Question

Is there a polynomial time algorithm for Boolean satisfiability?

CNF (Conjunctive Normal Form)

"and-of-or's"



$\varphi(x_1, \dots, x_n) =$

$(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\neg x_4) \wedge \dots$
"clause"

(CNF $\rightarrow$) SAT problem

Given boolean formula φ in CNF, decide if φ is satisfiable

More modest Question

Is there a polynomial time algorithm for CNF-satisfiability?

Pros

CNF formulas are even simpler, natural for computing
super easy to evaluate
CNF formulas

Cons

CNF formulas are also very expressive—universal?

too good to be true?

less sure about this

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

"SAT Programming Language"

★ "z = x" $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$ "both true or both false"

★ "if x then y else z" $(x \wedge y) \vee (\bar{x} \wedge z)$ "(x=1, y=1) or (x=0, z=1)"

Boolean Satisfiability

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$

• n variables • "size" m

e.g. $\varphi(0, 1, 0, 0, \dots, 1, 1) = 0$ (false) (true) (false)

Goal: decide if φ is "satisfiable";
i.e., if there exists $x_1, x_2, \dots, x_n \in \{0, 1\}$
s.t. $\varphi(x_1, x_2, \dots, x_n) = 1$

"search problem": we can recognize an answer (e.g. satisfying assignment x) when we see it (by evaluating $\varphi(x)$).

• at the very least: we can try all x .
• harder "search problem" than "over/under"

The Big Question

Is there a polynomial time algorithm for Boolean satisfiability?

CNF (Conjunctive Normal Form)

"and-of-or's"

$f(x_1, \dots, x_n) =$

$(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\neg x_4) \wedge \dots$

(CNF $\rightarrow$) SAT problem

Given boolean formula f in CNF, decide if f is satisfiable



More modest Question

Is there a polynomial time algorithm for CNF-satisfiability?

we don't know this either.

But we do know:

Theorem: there is a poly-time algo for Boolean SAT iff there is a poly-time algo for CNF-SAT.

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

"SAT Programming Language"

★ "z=x" $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$ "both true or both false"

★ "if x then y else z" $(x \wedge y) \vee (\bar{x} \wedge z)$ "(x=1, y=1) or (x=0, z=1)"

Boolean Satisfiability

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$
• n variables • "size" m

e.g. $\varphi(0, 1, 0, 0, \dots, 1, 1) = 0$ (false) (true) (false)

Goal: decide if φ is "satisfiable";
i.e., if there exists $x_1, x_2, \dots, x_n \in \{0, 1\}$
s.t. $\varphi(x_1, x_2, \dots, x_n) = 1$

"search problem": we can recognize an answer (e.g. satisfying assignment x) when we see it (by evaluating $\varphi(x)$).
• at the very least: we can try all x .
• harder "search problem" than "over/under"

The Big Question

Is there a polynomial time algorithm for Boolean satisfiability?

CNF (Conjunctive Normal Form)

"and-of-or's"

$f(x_1, \dots, x_n) =$

$(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\neg v \neg) \wedge \dots$

(CNF $\rightarrow$) SAT problem

Given boolean formula f in CNF, decide if f is satisfiable



Theorem: There is a poly-time algo for Boolean SAT iff there is a poly-time algo for CNF-SAT.

Proof. obv. algo for Boolean SAT is algo for CNF-SAT.

how to argue that algo for CNF-SAT $\Rightarrow$ algo for Boolean SAT?

Lemma. There is a polytime algo that, given a boolean formula $f(x_1, \dots, x_n)$ of size m , computes a CNF $g(x_1, \dots, x_p)$, w/ $p = O(mn)$ variables and size $O(m)$, s.t. f is satisfiable $\Leftrightarrow g$ is satisfiable

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$
 "x or y" $x \vee y$
 "not x" $\bar{x}$ (or $\neg x$)

"SAT Programming Language"

★ "z = x" $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$ "both true or both false"
 ★ "if x then y else z" $(x \wedge y) \vee (\bar{x} \wedge z)$ "(x=1, y=1) or (x=0, z=1)"

Boolean Satisfiability

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$
 • n variables • "size" m

e.g. $\varphi(0, 1, 0, 0, \dots, 1, 1) = 0$ (false) (true) (false)

Goal: decide if φ is "satisfiable";
 i.e., if there exists $x_1, x_2, \dots, x_n \in \{0, 1\}$
 s.t. $\varphi(x_1, x_2, \dots, x_n) = 1$

"search problem": we can recognize an answer (e.g. satisfying assignment x) when we see it (by evaluating $\varphi(x)$).
 • at the very least: we can try all x .
 • harder "search problem" than "over/under"

The Big Question

Is there a polynomial time algorithm for Boolean satisfiability?

CNF (Conjunctive Normal Form)

"and-of-or's" 

$f(x_1, \dots, x_n) = (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\dots) \wedge \dots$
"clause"

(CNF $\rightarrow$) SAT problem

Given boolean formula f in CNF, decide if f is satisfiable

Lemma. There is a polytime algo that, given a boolean formula $f(x_1, \dots, x_n)$ of size m , computes a CNF $g(x_1, \dots, x_p)$, w/ $p = O(m \log n)$ variables and size $O(m)$, s.t.
 f is satisfiable $\Leftrightarrow g$ is satisfiable

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

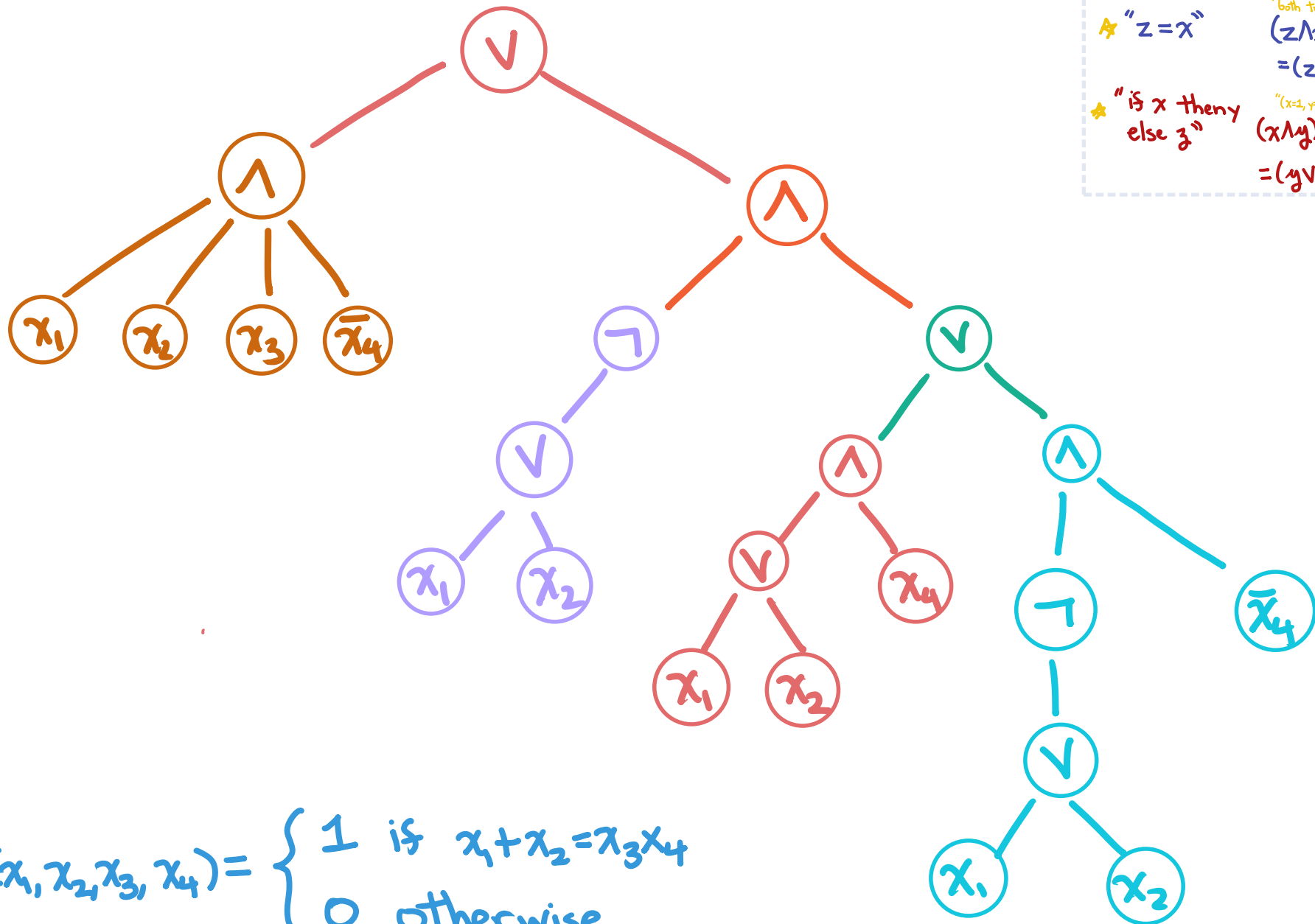
"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

"CNF-SAT Programming Language"

★ "z = x" ^{"both true or both false"}
 $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$
 $= (z \vee \bar{x}) \wedge (x \vee \bar{z})$

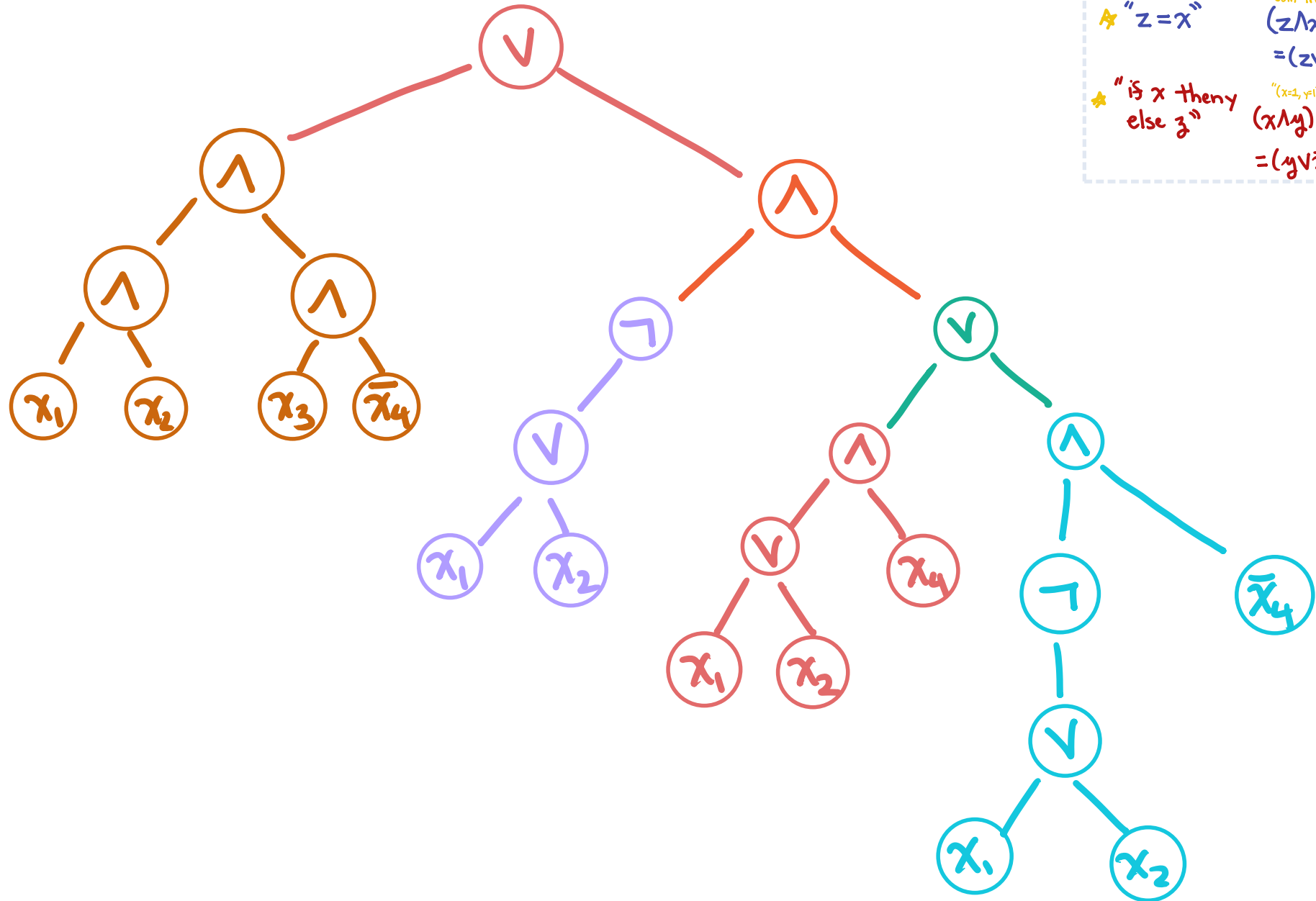
★ "if x then y else z"
^{"(x=1, y=1) or (x=0, z=1)"}
 $(x \wedge y) \vee (\bar{x} \wedge z)$
 $= (y \vee \bar{x}) \wedge (x \vee z) \wedge (x \vee \bar{z})$



$$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if } x_1 + x_2 = x_3 x_4 \\ 0 & \text{otherwise} \end{cases}$$

Preprocessing step:

convert to "binary" boolean formula



Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

"CNF-SAT Programming Language"

★ "z = x" ^{"both true or both false"}
 $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$
 $= (z \vee \bar{x}) \wedge (x \vee \bar{z})$

★ "if x then y else z" ^{"(x=1, y=1) or (x=0, z=1)"}
 $(x \wedge y) \vee (\bar{x} \wedge z)$
 $= (y \vee \bar{x}) \wedge (x \vee z) \wedge (x \vee \bar{z})$

Key idea: "assignment"

"CNF-SAT Programming Language"

"x and y"

$$x \wedge y$$

"x or y"

$$x \vee y$$

"not x"

$$\bar{x} \quad (\text{or } \neg x)$$

★ "z = x"

"both true or both false"

$$(z \wedge x) \vee (\bar{z} \wedge \bar{x})$$

$$= (z \vee (\bar{z} \wedge \bar{x})) \wedge (x \vee (\bar{z} \wedge \bar{x}))$$

$$= (z \vee \bar{z}) \wedge (z \vee \bar{x}) \wedge (x \vee \bar{z}) \wedge (x \vee \bar{x})$$

$$= (z \vee \bar{x}) \wedge (x \vee \bar{z})$$

★ "if x then y else z"

"(x=1, y=1) or (x=0, z=1)"

$$(x \wedge y) \vee (\bar{x} \wedge z)$$
$$= (x \vee \bar{x}) \wedge (y \vee \bar{x}) \wedge (x \vee z) \wedge (x \vee \bar{z})$$
$$= (y \vee \bar{x}) \wedge (x \vee z) \wedge (x \vee \bar{z})$$

So CNF is also very expressive

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

"SAT Programming Language"

★ "z = x" $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$

★ "if x then y else z" $(x \wedge y) \vee (\bar{x} \wedge z)$

Boolean Satisfiability

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$

• n variables • "size" m

e.g. $\varphi(0, 1, 0, 0, \dots, 1, 1) = 0$

Goal: decide if φ is "satisfiable";
i.e., if there exists $x_1, x_2, \dots, x_n \in \{0, 1\}$
s.t. $\varphi(x_1, x_2, \dots, x_n) = 1$

"search problem": we can recognize an answer (e.g. satisfying assignment x) when we see it (by evaluating $\varphi(x)$).
• at the very least: we can try all x .
• harder "search problem" than "over/under"

The Big Question

Is there a polynomial time algorithm for Boolean satisfiability?

CNF (Conjunctive Normal Form)

"and-of-or's"

$\varphi(x_1, \dots, x_n) =$

$(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\neg x_4) \wedge \dots$

(CNF $\rightarrow$) SAT problem

Given boolean formula φ in CNF, decide if φ is satisfiable

$$x_1 = (x_2 \wedge x_3)$$

$$x_1 = (x_2 \vee x_3)$$

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

"CNF-SAT Programming Language"

★ "z = x"

"both true or both false"

$$(z \wedge x) \vee (\bar{z} \wedge \bar{x})$$

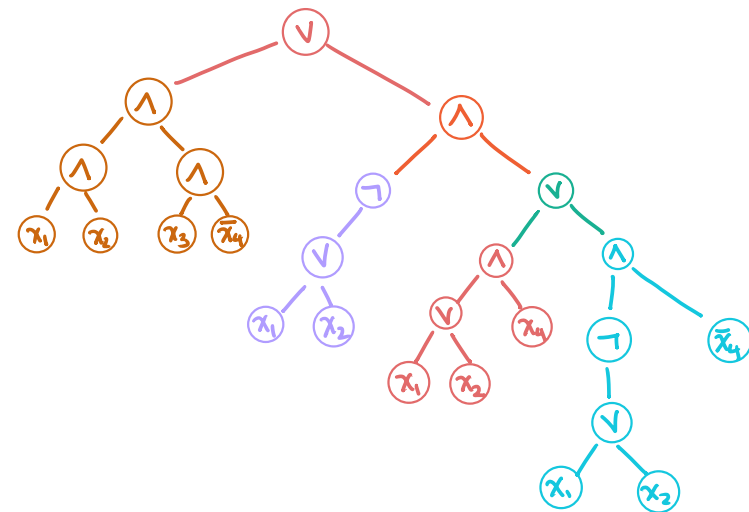
$$= (z \vee \bar{x}) \wedge (x \vee \bar{z})$$

★ "if x then y else z"

"(x=1, y=1) or (x=0, z=1)"

$$(x \wedge y) \vee (\bar{x} \wedge z)$$

$$= (y \vee \bar{x}) \wedge (x \vee z) \wedge (x \vee \bar{z})$$



Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

" x or y " $x \vee y$

"not x " $\bar{x}$ (or $\neg x$)

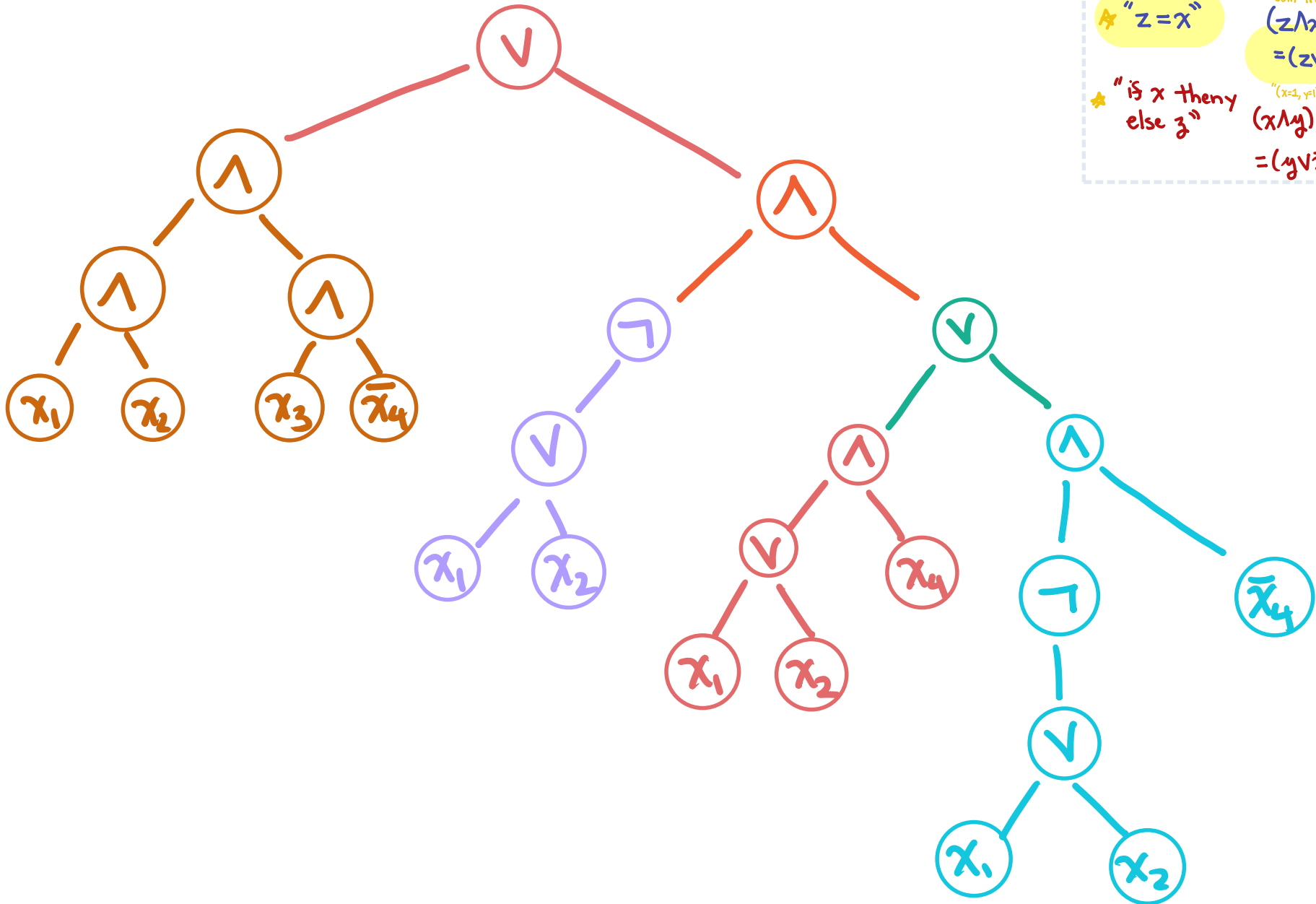
"CNF-SAT Programming Language"

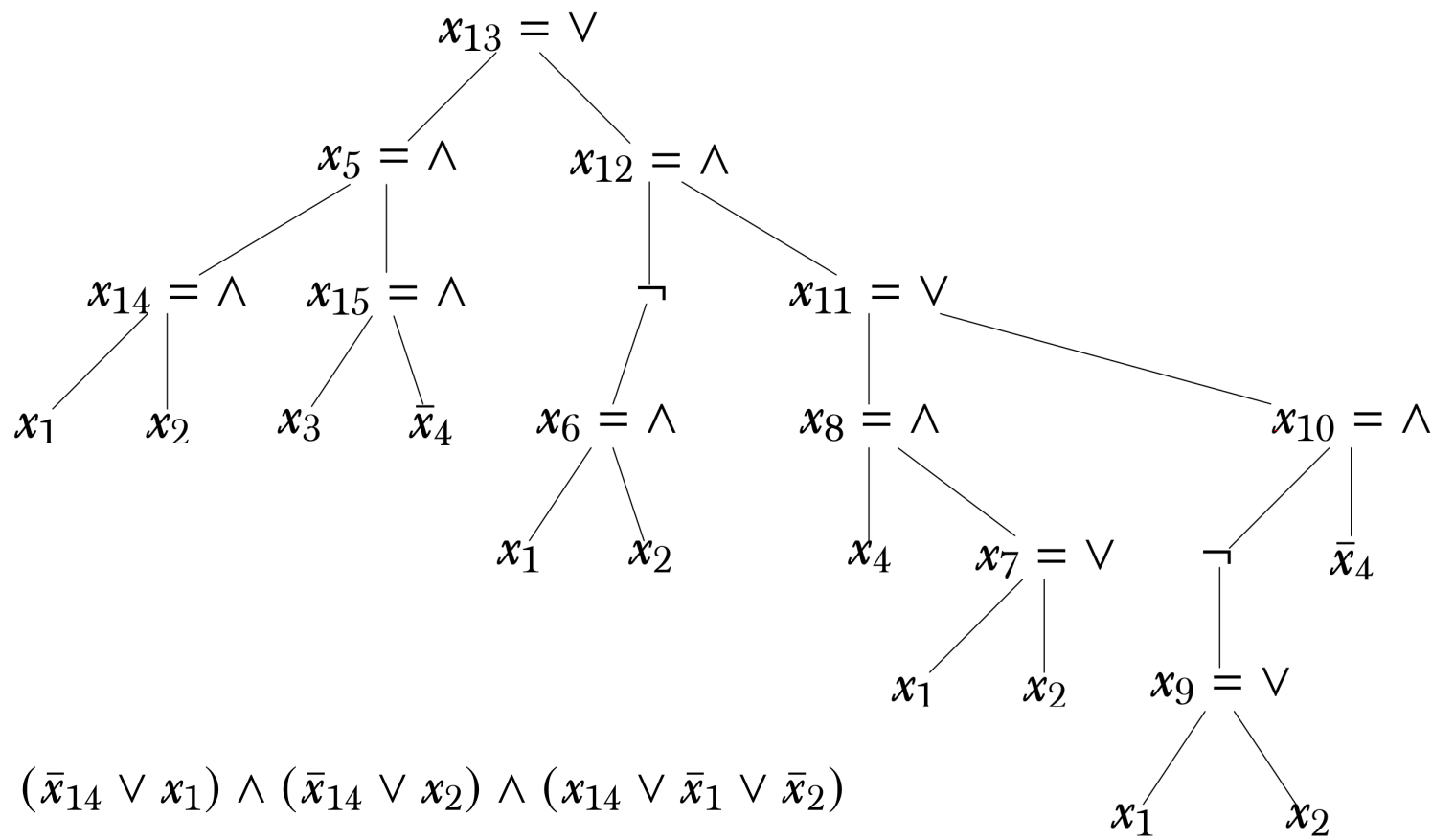
A "z=x"

"both true or both false"

$$(z \wedge x) \vee (\bar{z} \wedge \bar{x})$$
$$= (z \vee \bar{x}) \wedge (x \vee \bar{y})$$

★ "if x then y
else z"

$$"(x=1, y=1) \text{ or } (x=0, y=1)"$$
$$(x \wedge y) \vee (\bar{x} \wedge z)$$
$$= (y \vee \bar{x}) \wedge (x \vee z) \wedge (x \vee \bar{z})$$




Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

"CNF-SAT Programming Language"

★ "z = x"

"both true or both false"

$(z \wedge x) \vee (\bar{z} \wedge \bar{x})$

$=(z \vee \bar{x}) \wedge (x \vee \bar{z})$

★ "if x then y else z"

"(x=1, y=1) or (x=0, y=1)"

$(x \wedge y) \vee (\bar{x} \wedge z)$

$=(y \vee \bar{x}) \wedge (x \vee z) \wedge (x \vee \bar{z})$

$$(\bar{x}_{14} \vee x_1) \wedge (\bar{x}_{14} \vee x_2) \wedge (x_{14} \vee \bar{x}_1 \vee \bar{x}_2)$$

$$(\bar{x}_{15} \vee x_3) \wedge (\bar{x}_{15} \vee \bar{x}_4) \wedge (x_{15} \vee \bar{x}_3 \vee x_4)$$

$$(\bar{x}_6 \vee x_1) \wedge (\bar{x}_6 \vee x_2) \wedge (x_6 \vee \bar{x}_1 \vee \bar{x}_2)$$

$$(x_7 \vee \bar{x}_1) \wedge (x_7 \vee \bar{x}_2) \wedge (\bar{x}_7 \vee x_1 \vee x_2)$$

$$(\bar{x}_8 \vee x_4) \wedge (\bar{x}_8 \vee x_7) \wedge (x_8 \vee \bar{x}_4 \vee \bar{x}_7)$$

$$(x_9 \vee \bar{x}_1) \wedge (x_9 \vee \bar{x}_2) \wedge (\bar{x}_9 \vee x_1 \vee x_2)$$

$$(\bar{x}_{10} \vee \bar{x}_9) \wedge (\bar{x}_{10} \vee \bar{x}_4) \wedge (x_{10} \vee x_9 \vee x_4)$$

$$(x_{11} \vee \bar{x}_8) \wedge (x_{11} \vee \bar{x}_{10}) \wedge (\bar{x}_{11} \vee x_8 \vee x_{10})$$

$$(\bar{x}_{12} \vee \bar{x}_6) \wedge (\bar{x}_{12} \vee x_{11}) \wedge (x_{12} \vee x_6 \vee \bar{x}_{11})$$

$$(x_{13} \vee \bar{x}_5) \wedge (x_{13} \vee \bar{x}_{12}) \wedge (\bar{x}_{13} \vee x_5 \vee x_{12})$$

✓ Theorem: There is a poly-time algo for Boolean SAT iff there is a poly-time algo for CNF-SAT.

Proof. obv. algo for Boolean SAT is algo for CNF-SAT.

how to argue that algo for CNF-SAT $\Rightarrow$ algo for Boolean SAT?

✓ Lemma. There is a polytime algo that, given a boolean formula $f(x_1, \dots, x_n)$ of size m , computes a CNF $g(x_1, \dots, x_p)$, w/ $p = O(mn)$ variables and size $O(m)$, s.t. f is satisfiable $\Leftrightarrow g$ is satisfiable

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$
 "x or y" $x \vee y$
 "not x" $\bar{x}$ (or $\neg x$)

"SAT Programming Language"

★ "z = x" $(z \wedge x) \vee (\bar{z} \wedge \bar{x})$ "both true or both false"
 ★ "if x then y else z" $(x \wedge y) \vee (\bar{x} \wedge z)$ "(x=1, y=1) or (x=0, z=1)"

Boolean Satisfiability

Input: Boolean formula $\varphi(x_1, x_2, \dots, x_n)$
 • n variables • "size" m

e.g. $\varphi(0, 1, 0, 0, \dots, 1, 1) = 0$ (false) (true) (false)

Goal: decide if φ is "satisfiable";
 i.e., if there exists $x_1, x_2, \dots, x_n \in \{0, 1\}$
 s.t. $\varphi(x_1, x_2, \dots, x_n) = 1$

"search problem": we can recognize an answer (e.g. satisfying assignment x) when we see it (by evaluating $\varphi(x)$).
 • at the very least: we can try all x .
 • harder "search problem" than "over/under"

The Big Question

Is there a polynomial time algorithm for Boolean satisfiability?

CNF (Conjunctive Normal Form)

"and-of-or's" 

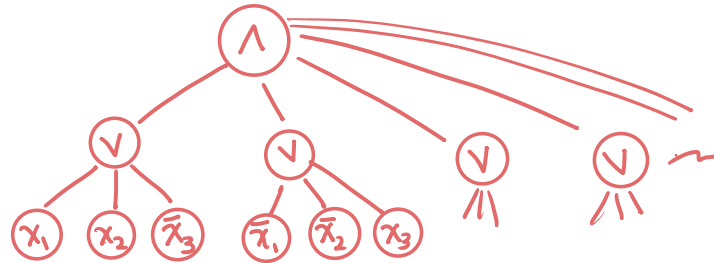
$f(x_1, \dots, x_n) = (\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\dots) \wedge \dots$
"clause"

(CNF $\rightarrow$) SAT problem

Given boolean formula f in CNF, decide if f is satisfiable

more special case: 3-SAT

like CNF-SAT, except w/ exactly 3 variables per clause.



$$f(x_1, \dots, x_n) =$$

$$(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\sim \vee \sim \vee \sim) \wedge \sim$$

"clause"

Is there a polytime algo for 3SAT?

Theorem: there is a poly-time algo for Boolean SAT iff there is a poly-time algo for 3-SAT.

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

CNF (Conjunctive Normal Form)

"and-of-or's"



$$f(x_1, \dots, x_n) =$$

$$(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\sim \vee \sim \vee \sim) \wedge \sim$$

"clause"

(CNF $\rightarrow$) SAT problem

Given boolean formula f in CNF, decide if f is satisfiable

"CNF-SAT Programming Language"

★ "z = x"

"both true or both false"

$$(z \wedge x) \vee (\bar{z} \wedge \bar{x})$$

$$= (z \vee \bar{x}) \wedge (x \vee \bar{z})$$

★ "if x then y else z"

"(x=1, y=1) or (x=0, y=0)"

$$(x \wedge y) \vee (\bar{x} \wedge \bar{y})$$

$$= (y \vee \bar{x}) \wedge (x \vee \bar{y})$$

Theorem: there is a poly-time algo for Boolean SAT iff there is a poly-time algo for CNF-SAT.

Lemma. There is a polytime algo that, given a boolean formula $f(x_1, \dots, x_n)$ of size m , computes a CNF $g(x_1, \dots, x_p)$, w/ $p = O(m \log n)$ variables and size $O(m)$, s.t.

$$f \text{ is satisfiable} \Leftrightarrow g \text{ is satisfiable}$$

Theorem: there is a poly-time algo for Boolean SAT iff there is a poly-time algo for 3-SAT.

how to prove this?

show how to convert any CNF to 3-SAT w/ similar total size

same proof ideas as before.

Boolean logic $x, y \in \{\text{TRUE}, \text{FALSE}\} = \{1, 0\}$

"x and y" $x \wedge y$

"x or y" $x \vee y$

"not x" $\bar{x}$ (or $\neg x$)

CNF (Conjunctive Normal Form)

"and-of-or's"



$f(x_1, \dots, x_n) =$

$(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (\neg x_4) \wedge \dots$

(CNF $\rightarrow$) SAT problem

Given boolean formula f in CNF, decide if f is satisfiable

"CNF-SAT Programming Language"

★ "z = x"

"both true or both false"

$(z \wedge x) \vee (\bar{z} \wedge \bar{x})$

$= (z \vee \bar{x}) \wedge (x \vee \bar{z})$

★ "if x then y else z"

"(x=1, y=1) or (x=0, y=0)"

$(x \wedge y) \vee (\bar{x} \wedge \bar{y})$

$= (y \vee \bar{x}) \wedge (x \vee \bar{y})$

Theorem: there is a poly-time algo for Boolean SAT iff there is a poly-time algo for CNF-SAT.

Lemma. There is a polytime algo that, given a boolean formula $f(x_1, \dots, x_n)$ of size m , computes a CNF $g(x_1, \dots, x_p)$, w/ $p = O(m \log n)$ variables and size $O(m)$, s.t.

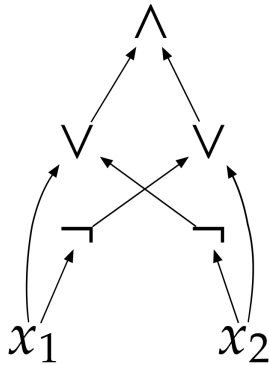
f is satisfiable $\Leftrightarrow g$ is satisfiable

3-SAT. like CNF-SAT, except w/ exactly 3 variables per clause.

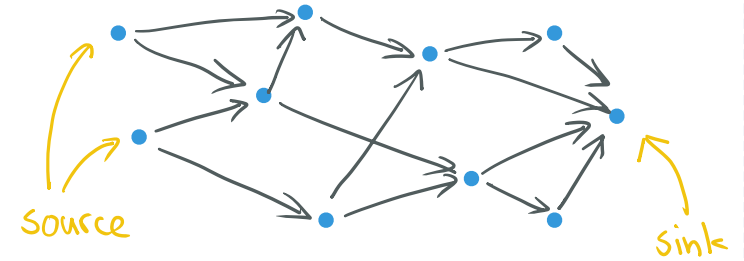
e.g. circuits

XOR ("exclusive-or")

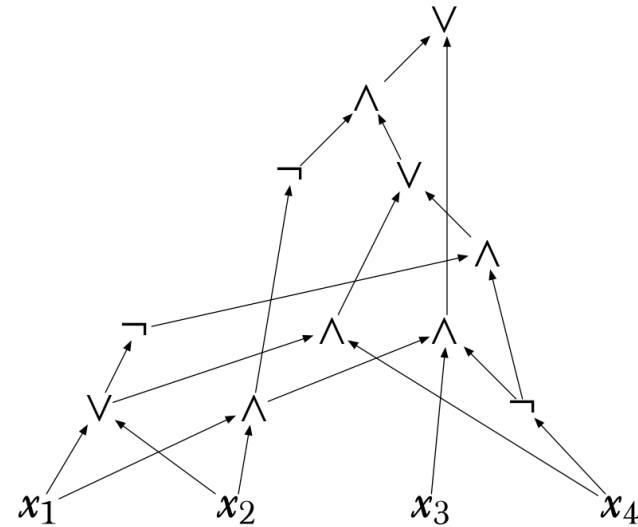
(TRUE iff $x_1 \neq x_2$)



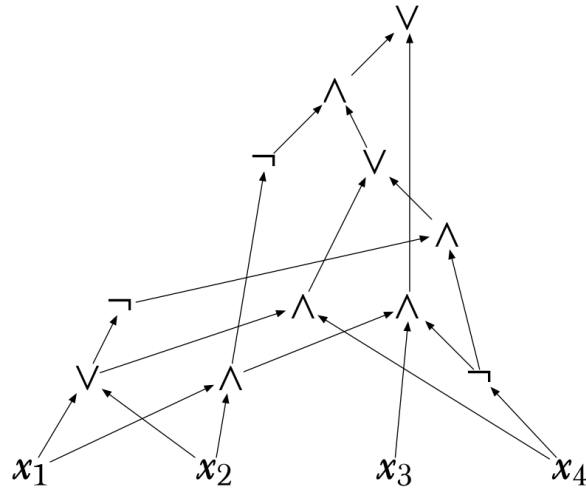
$$f(x_1, x_2) = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$$



Addition



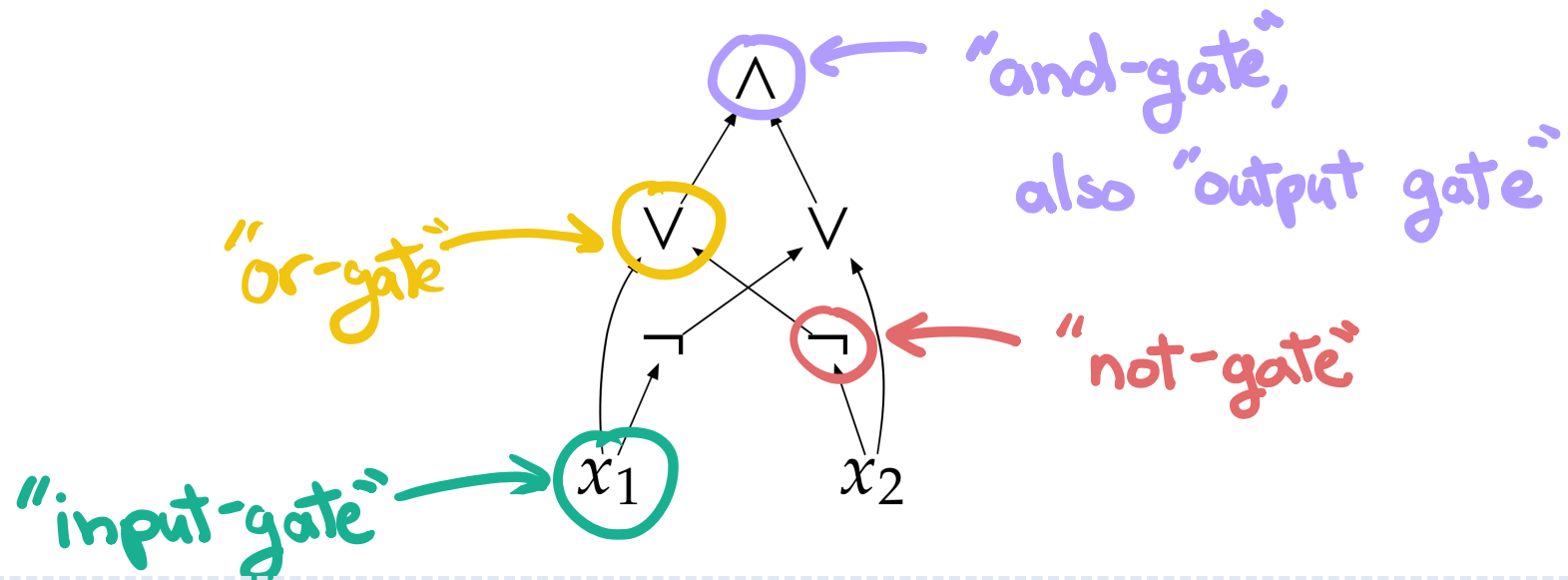
$$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if } x_1 + x_2 = x_3 x_4 \\ 0 & \text{otherwise} \end{cases}$$



(definition)

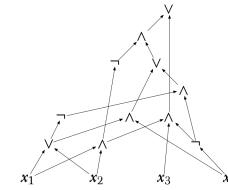
logical gates arranged as vertices
in a directed acyclic graph

"gates": take some inputs bits, output 1 bit



Circuit SAT

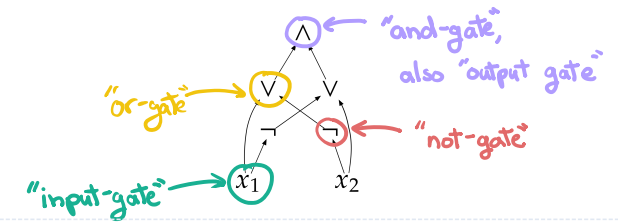
given a circuit, decide if it is satisfiable



(definition)

logical gates arranged as vertices in a directed acyclic graph

"gates": take some inputs bits, output 1 bit



Theorem: there is a poly-time algo for Boolean SAT iff there is a poly-time algo for Circuit SAT

Conclusion

We want, but don't have, polynomial time algos for

Boolean SAT, CNF SAT, 3-SAT, or Circuit SAT

But we do know they are equivalent up to polynomial time!