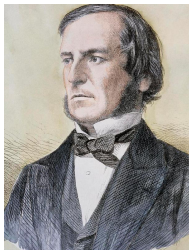


Subset Sum, and the  
secret language of numbers

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  $\neg x$ )



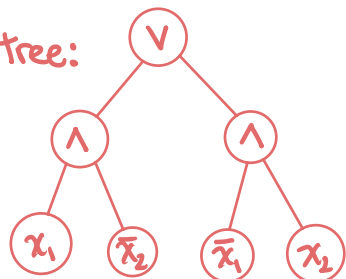
George Boole,  
1815-1864

XOR ("exclusive-or")

(TRUE iff  $x_1 \neq x_2$ )

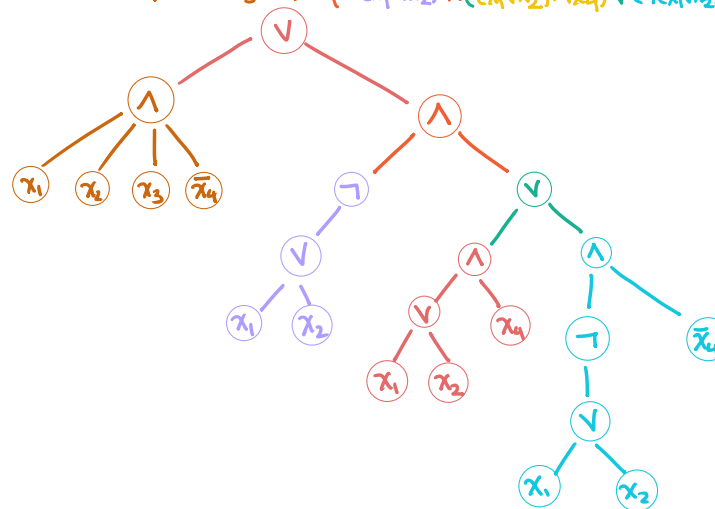
$$f(x_1, x_2) = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$$

as a tree:



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

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



"SAT Programming Language"

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

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

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

So Boolean algebra is very expressive

Boolean Satisfiability

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

• n variables • "size" m

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

Goal: decide if  $\varphi$  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 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  $x_1 \neq x_2$ )

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



Goal: The Big Question

Is there a polynomial time algorithm for Boolean satisfiability?

$x_2$

$x_2 = x_3 x_4$

wise

$x_3 \wedge \bar{x}_4$

$x_2$

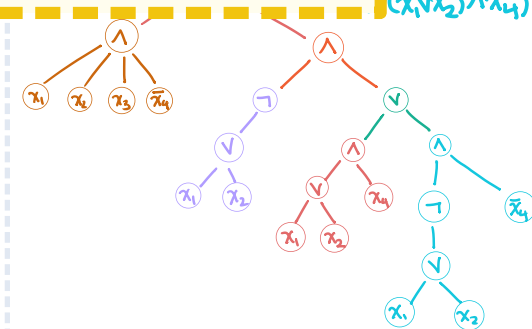
$x_2 \wedge x_4$

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

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"

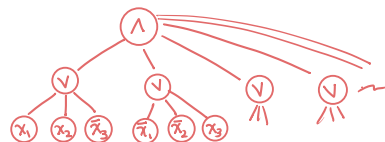
"(x=1, y=1) or (x=0, z=1)"  
 $(x \wedge y) \vee (\bar{x} \wedge z)$

## CNF (Conjunctive Normal Form)

"and-of-or's"

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

$$(\bar{x}_1 \vee x_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee x_2) \wedge \dots$$



(CNF-) 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?

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

## "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"

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

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

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

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

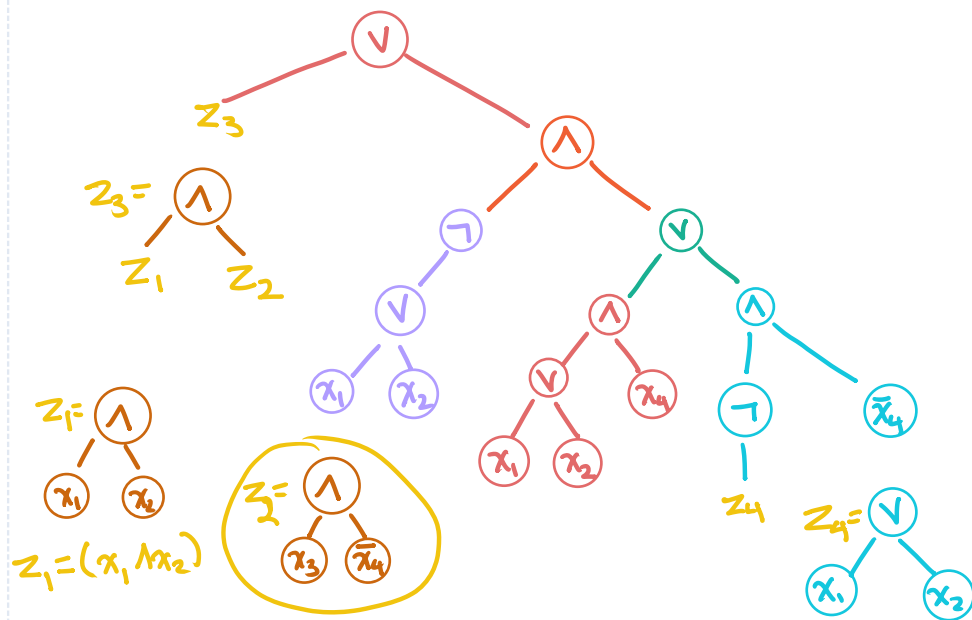
★ "if x then y else z"

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

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

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

So CNF is also very expressive

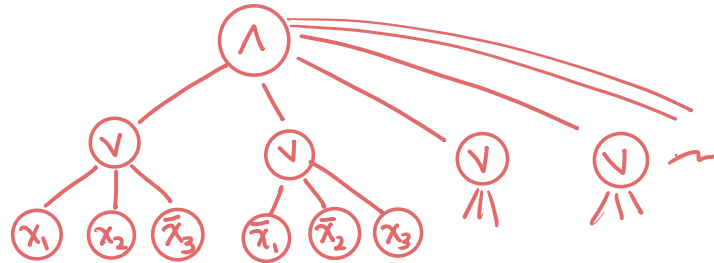




more special case:

## 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)"

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

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

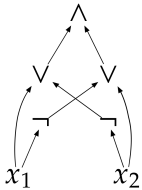
Theorem: there is a poly-time algo for Boolean SAT iff there is a poly-time algo for CNF-SAT.

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.

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

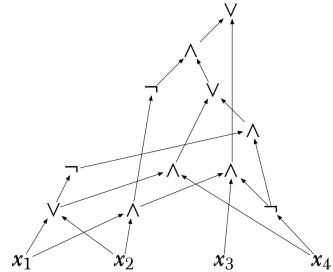
e.g. circuits

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

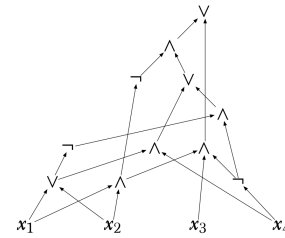
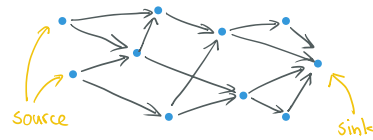


$$f(x_1, x_2) = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$$

Addition



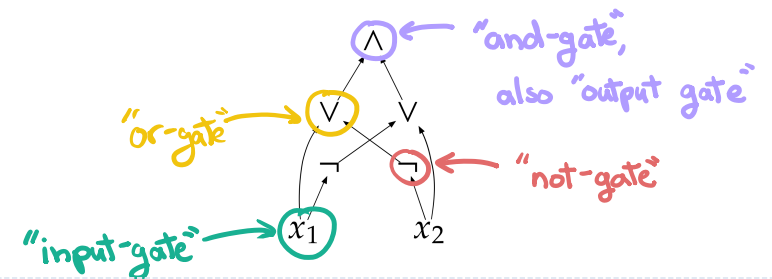
$$f(x_1, x_2, x_3, x_4) = \begin{cases} 1 & \text{if } x_1 + x_2 = x_3 x_4 \\ 0 & \text{otherwise} \end{cases}$$



(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

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

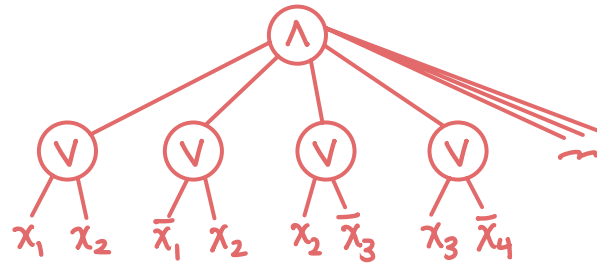
But we do know they are equivalent up to polynomial time!

# 2-SAT

like CNF-SAT, except w/ exactly 2 variables per clause.

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

$$(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2) \wedge \underbrace{(x_2 \vee \bar{x}_3)}_{\text{clause}} \wedge (x_3 \vee \bar{x}_4) \wedge \dots$$



## Max 2-SAT

like 2-SAT, except goal is to satisfy  
maximum number of clauses.  
(not necessarily all)

Is there a polytime algo for Max 2-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"



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

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

(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/  
 $p = O(mn)$  variables and size  $O(m)$ , s.t.

$f$  is satisfiable  $\Leftrightarrow g$  is satisfiable

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

|           | Polytime Algo | Hard as <sup>(CNF-)</sup> SAT | Abstain |
|-----------|---------------|-------------------------------|---------|
| 2SAT      | 66%           | 44%                           | 2       |
| Max 2-SAT | 60%           | 35%                           | 5       |

let  $C$  be a sink SCC.

Complement  $\bar{C}$  is a source SCC (why?)



Assign (all literals in)  $C = \text{true}$ ,  $\bar{C} = \text{false}$

Remove  $C$  and  $\bar{C}$  and recurse on remaining SCC's

Implication graph

- vertex for each  $x_i, \bar{x}_i$

- for clause  $(a \vee b)$ ,

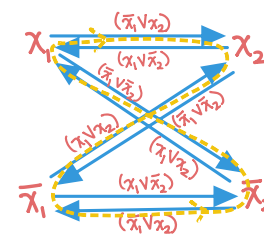
$$\bar{a} \xrightarrow{(a \vee b)} b \quad \bar{b} \xrightarrow{(a \vee b)} a$$

" $\bar{a} = \text{TRUE} \Rightarrow b = \text{TRUE}$ "

" $\bar{b} = \text{TRUE} \Rightarrow a = \text{TRUE}$ "

$$\mathcal{I}(x_1, x_2) =$$

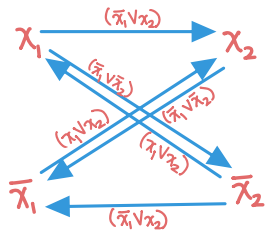
$$(x_1 \vee x_2) \wedge (\bar{x}_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee \bar{x}_2)$$



directed cycle w/  $x_i$  &  $\bar{x}_i$

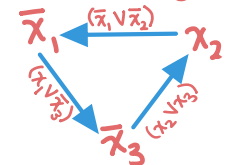
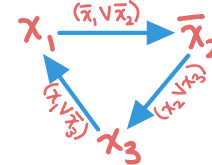
$$\mathcal{I}(x_1, x_2) =$$

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



no directed cycle w/ complementary literals

all literals in SCC have same assignment



each SCC  $C$  has complement SCC  $\bar{C}$  w/ negated literals & reversed edges

Key idea: assigning sinks to be true,  
sources to be false  
does not violate any edges

Fix a clause  $(a \vee b \vee c)$

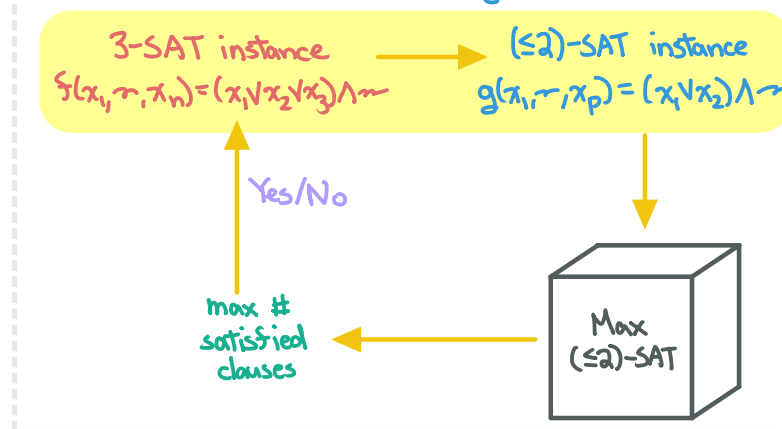
introduce new variable  $y$

consider the 10 clauses

$(a), (b), (c), (\bar{a} \vee \bar{b}), (\bar{b} \vee \bar{c}), (\bar{c} \vee \bar{a}),$

$(y), (a \vee \bar{y}), (b \vee \bar{y}), (c \vee \bar{y})$

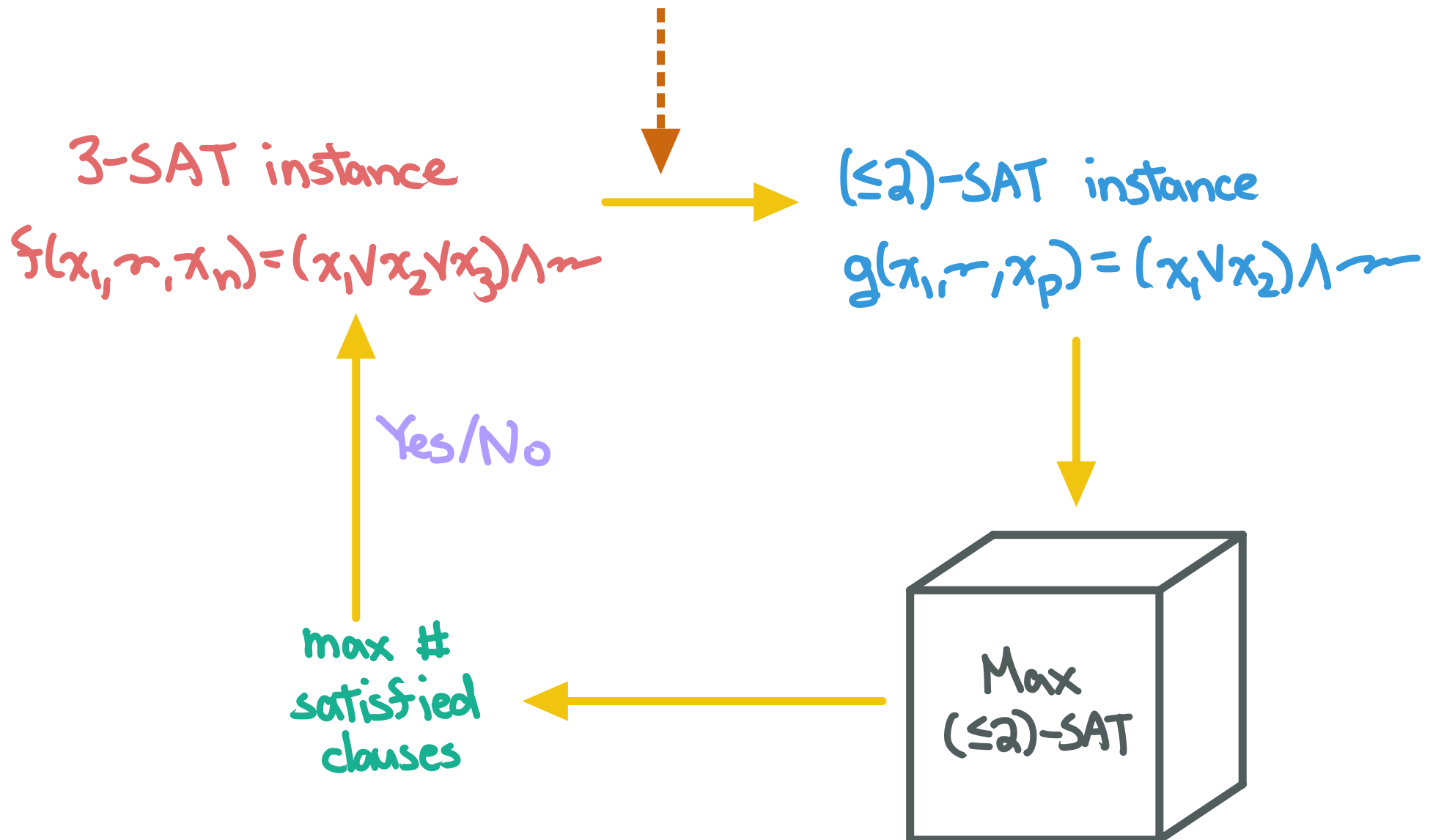
Theorem: A polytime algorithm for max  $(\leq 2)$ -SAT implies a polynomial time algorithm for SAT.



| $(a, b, c) =$                                                                     | $(f, f, f)$ | $(t, f, f)$ | $(t, t, f)$ | $(t, t, t)$ |
|-----------------------------------------------------------------------------------|-------------|-------------|-------------|-------------|
| $(a), (b), (c)$                                                                   | 0           | 1           | 2           | 3           |
| $(\bar{a} \vee \bar{b}), (\bar{b} \vee \bar{c}), (\bar{c} \vee \bar{a})$          | 3           | 3           | 2           | 0           |
| $\max_{y \in \{t, f\}} (y), (a \vee \bar{y}), (b \vee \bar{y}), (c \vee \bar{y})$ | 3           | 3           | 3           | 4           |
| <b>total</b>                                                                      | 6           | 7           | 7           | 7           |

**Lemma 10.3.** Given a disjunction of 3 literals,  $C = (a \vee b \vee c)$ , one can compute a family  $\mathcal{F}_C$  of 10 disjunctive clauses over  $a, b, c$  and an auxiliary variable  $y$ , each with at most 2 literals, such that for any assignment to  $a, b, c$ :

- (a) At most 7 clauses in  $\mathcal{F}_C$  can be satisfied.
- (b) If  $(a \vee b \vee c) = \text{true}$ , then there is an assignment  $y \in \{\text{true}, \text{false}\}$  that satisfies 7 of the clauses in  $\mathcal{F}_C$ .
- (c) If  $(a \vee b \vee c) = \text{false}$ , then any choice of  $y \in \{\text{true}, \text{false}\}$  satisfies at most 6 of the clauses in  $\mathcal{F}_C$ .





Theorem: A polytime algorithm for max 2-SAT implies a polynomial time algorithm for SAT.

$$(a_i \vee y_i) \wedge (a_i \vee \bar{y}_i)$$

$(\leq 2)$ -SAT instance

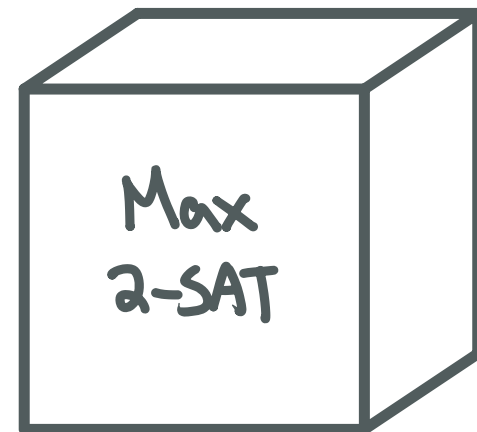
$$f(x_1, \dots, x_n) = (x_1 \vee x_2 \vee x_3) \wedge \dots$$

2-SAT instance

$$g(x_1, \dots, x_p) = (x_1 \vee x_2) \wedge \dots$$

max #  
satisfied  
clauses in  $g$

$-k$



e.g. Making change (\$)

93 cents = ..... quarters,

..... dimes,

..... nickels,

..... pennies

# Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: Find a subset of  $x_i$ 's that sum to  $T$   
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10

$T=52$

Another "search" problem

|            | Polytime Algo | Hard as <sup>(CNF-)</sup> SAT | Abstain |
|------------|---------------|-------------------------------|---------|
| Subset Sum |               |                               |         |

# Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: Find a subset of  $x_i$ 's that sum to  $T$   
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10

$T=52$

---

Ideas for algorithms?

# Recursive Spec

(Subset-Sum)

$SS((x_1, \dots, x_n), T) =$

TRUE if there is a subset of  $x_1, \dots, x_n$  summing to  $T$ ,  
and FALSE otherwise

Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: Find a subset of  $x_i$ 's that sum to  $T$   
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10  
 $T=52$

## Implementation

$SS((x_1, \dots, x_n), T)$

1. If  $T < 0$  then return FALSE

2. If  $T = 0$  then return TRUE

3. If  $n = 0$  then return FALSE

4. Return  $SS((x_2, \dots, x_n), T) \vee SS((x_2, \dots, x_n), T - x_1)$

Running time:  $T(n) = T(n-1) + O(1) = O(2^n)$

## Recursive Spec

(Subset-Sum)

$SS(i, S) = \text{TRUE}$  if there is a subset of  $x_1, \dots, x_n$  summing to  $S$ , and  $\text{FALSE}$  otherwise

Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: Find a subset of  $x_i$ 's that sum to  $T$   
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10  
 $T=52$

## Implementation

$SS(i, S)$

1. If  $S < 0$  then return  $\text{FALSE}$
2. If  $S = 0$  then return  $\text{TRUE}$
3. If  $n = 0$  then return  $\text{FALSE}$
4. Return  $SS(i+1, S) \vee SS(i+1, S - x_i)$

Running time:

$O(nT)$  subproblems

$O(1)$  time per subproblem

---

$O(nT)$  time overall

# Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: Find a subset of  $x_i$ 's that sum to  $T$   
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10

$T=52$

|            | Polytime Algo | Hard as <sup>(CNF-)</sup> SAT | Abstain |
|------------|---------------|-------------------------------|---------|
| Subset Sum |               |                               |         |

Running time:

$O(nT)$  subproblems

$O(1)$  time per subproblem

---

$O(nT)$  time overall

"T" takes  $\log(T)$  bits to write down

what if  $T \approx 2^n$ ?

Sh\*t. Any other ideas?

Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: Find a subset of  $x_i$ 's that sum to  $T$   
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10  
 $T=52$

Recursive Spec

(Subset-Sum)  
 $SS(i, S) = \text{TRUE}$  if there is a subset of  $x_1, \dots, x_n$  summing to  $S$ , FALSE otherwise

Implementation

$SS(i, S)$

1. If  $S < 0$  then return FALSE
2. If  $S = 0$  then return TRUE
3. If  $n = 0$  then return FALSE
4. Return  $SS(i+1, S) \vee SS(i+1, S - x_i)$

Running time:

$O(nT)$  subproblems

$O(1)$  time per subproblem

---

$O(nT)$  time overall



Can we instead show that  
Subset-Sum is hard as SAT?

Unlike previous problems, SubsetSum  
looks nothing like SAT ~

Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: Find a subset of  $x_i$ 's that sum to  $T$   
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10  
 $T=52$

Recursive Spec

(Subset-Sum)  
 $SS(i, S) = \text{TRUE}$  if there is a subset  
of  $x_1, \dots, x_n$  summing to  $S$ , FALSE otherwise

Implementation

$SS(i, S)$

1. If  $S < 0$  then return FALSE
2. If  $S = 0$  then return TRUE
3. If  $n = 0$  then return FALSE
4. Return  $SS(i+1, S) \vee SS(i+1, S - x_i)$

Running time:

$O(nT)$  subproblems

$O(1)$  time per subproblem

$O(nT)$  time overall

"T" takes  $\log(T)$  bits to write down

Theorem: A polytime algo for subset-sum  
implies a polytime algo for SAT

Theorem: A polytime algo for subset-sum implies a polytime algo for SAT

Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: Find a subset of  $x_i$ 's that sum to  $T$   
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10  
 $T=52$

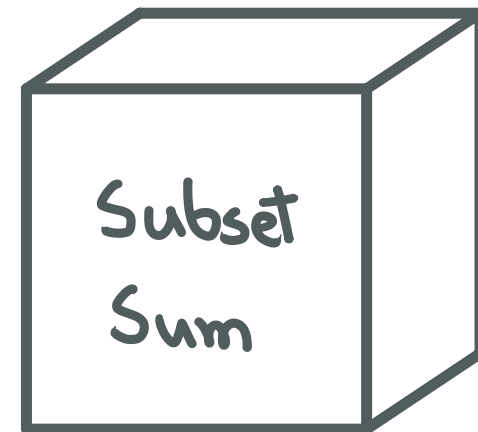
Proof setup.

3-SAT instance

$$\mathcal{F}(x_1, \dots, x_n) = (x_1 \vee x_2 \vee x_3) \wedge \dots$$

$x_1, x_2, \dots, x_p \in \mathbb{N}$   
 $T \in \mathbb{N}$

Yes/No for  
 $x_1, x_2, \dots, x_p \in \mathbb{N}$   
 $T \in \mathbb{N}$



Key idea: numbers are bit-strings too!

— use many digits

— each digit encodes 1 constraint

Key idea: numbers are bit-strings too!

• use many digits • each digit encodes 1 constraint

1. Recast 3SAT as "subset" problem

$X = \text{literals } \{x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n\}$

clause  $(x_1 \vee \bar{x}_2 \vee x_3) = \text{set } \{x_1, \bar{x}_2, x_3\} \subset X$

Goal: find  $S \subset X$  s.t.

Ⓐ for each clause  $C_i$ ,  $C_i \cap S = \emptyset$

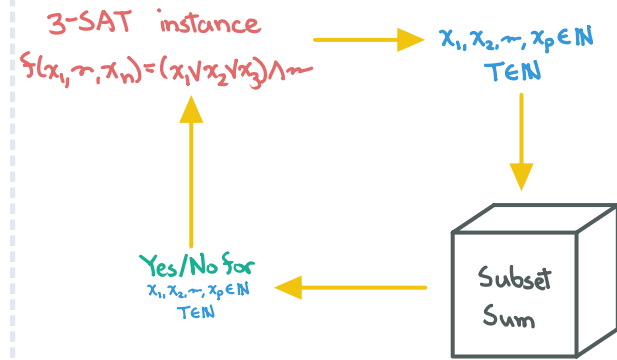
Ⓑ for each  $x_j$ ,  $S$  contains (exactly) 1 of  $\{x_j, \bar{x}_j\}$

Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: find a subset of  $x_i$ 's that sum to  $T$   
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10  
 $T=52$

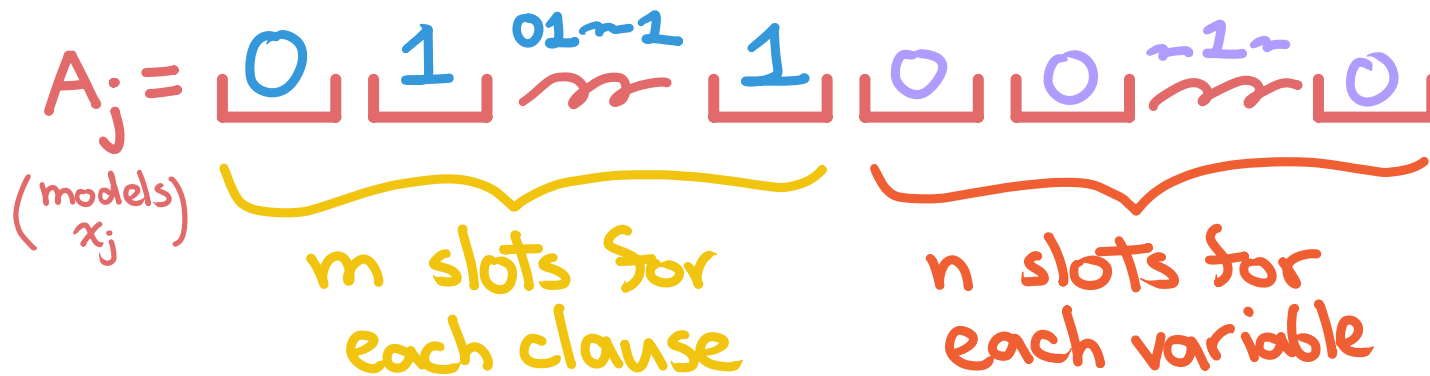


Key idea: numbers are bit-strings too!

- use many digits • each digit encodes 1 constraint

2. Start making #'s:

- we use mtn digits base 4
- for the literal  $x_j$ :



- for each clause  $C_i$ , set  $C_i$ th digit to 1 if  $x_j \in C_i$ , else 0

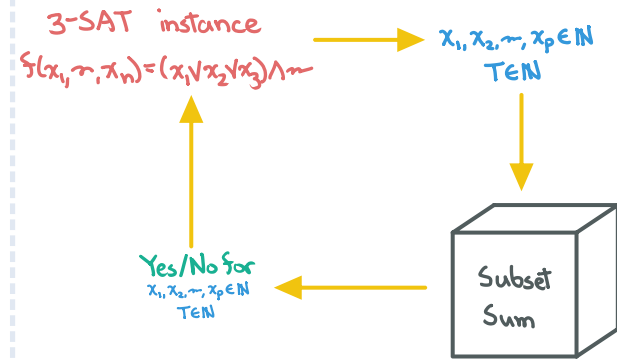
- set " $x_j$ th digit" to 1, else 0

Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: Find a subset of  $x_i$ 's that sum to  $T$   
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10  
 $T=52$



1. Recast 3SAT as "subset" problem  
 $X = \text{literals } \{x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n\}$

clause  $(x_1 \vee \bar{x}_2 \vee x_3) = \text{set } \{x_1, \bar{x}_2, x_3\} \subset X$

Goal: Find  $S \subset X$  s.t.

(a) for each clause  $C_i$ ,  $C_i \cap S = \emptyset$

(b) for each  $x_j$ ,  $S$  contains <sup>(exactly)</sup> 1 of  $\{x_j, \bar{x}_j\}$

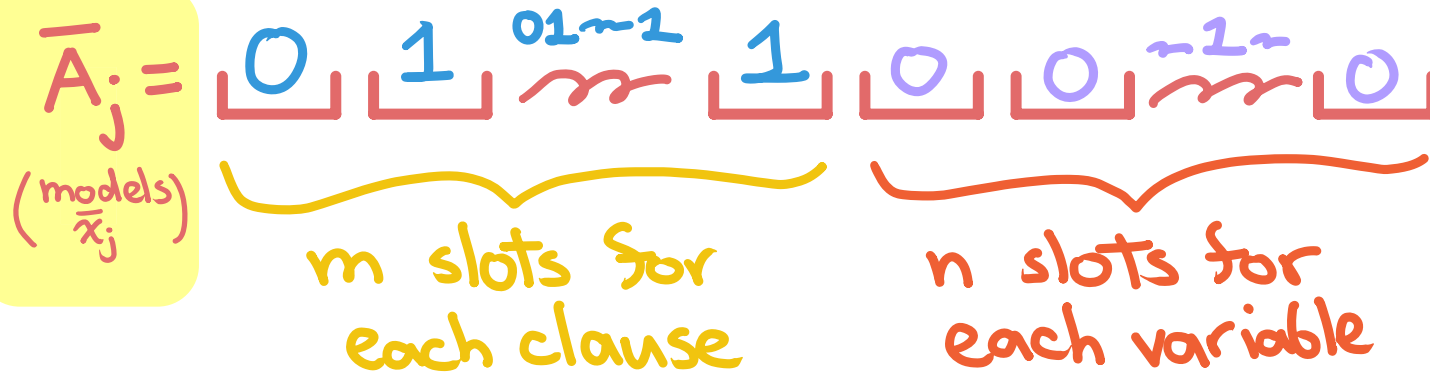
(not  $\bar{x}_j$ )

Key idea: numbers are bit-strings too!

- use many digits • each digit encodes 1 constraint

2. Start making #'s:

- we use mtn digits base 4
- Similarly, for the literal  $\bar{x}_j$ :



- for each clause  $C_i$ , set  $C_i$ th digit to 1 if  $\bar{x}_j \in C_i$ , else 0

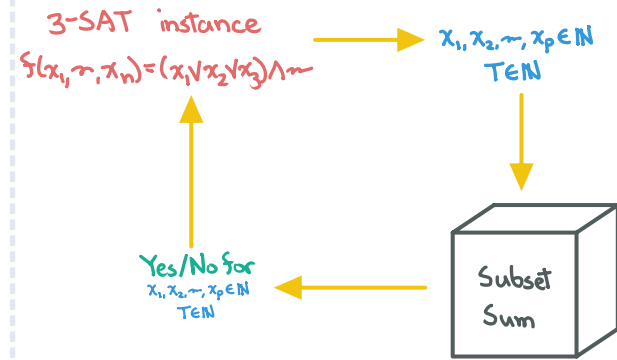
- set " $x_j$ th digit" to 1, else 0

Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: Find a subset of  $x_i$ 's that sum to T  
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10  
T=52



1. Recast 3SAT as "subset" problem

$X = \text{literals } \{x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n\}$

clause  $(x_1 \vee \bar{x}_2 \vee x_3) = \text{set } \{x_1, \bar{x}_2, x_3\} \subset X$

Goal: Find  $S \subset X$  s.t.

(a) for each clause  $C_i$ ,  $C_i \cap S \neq \emptyset$

(b) for each  $x_j$ ,  $S$  contains exactly 1 of  $\{x_j, \bar{x}_j\}$

(not  $x_j$ )

$\bar{x}_j \in C_i$

Key idea: numbers are bit-strings too!

• use many digits • each digit encodes 1 constraint

Literals

Long #'s

$$X = \{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\} \leftrightarrow \{A_1, \bar{A}_1, \dots, A_n, \bar{A}_n\}$$

Set SCX

$$D = \sum_{x_j \in S} A_j + \sum_{\bar{x}_j \in S} \bar{A}_j$$

$$\forall \text{ constraint } C_i, \quad C_i \cap S \neq \emptyset \quad \leftrightarrow \quad \forall \text{ constraint } C_i, \quad C_i \text{th digit of } D \geq 1$$

$$\forall \text{ variables } x_j \quad \leftrightarrow \quad \forall \text{ variables } x_j, \quad |\{x_j, \bar{x}_j\} \cap S| = 1$$

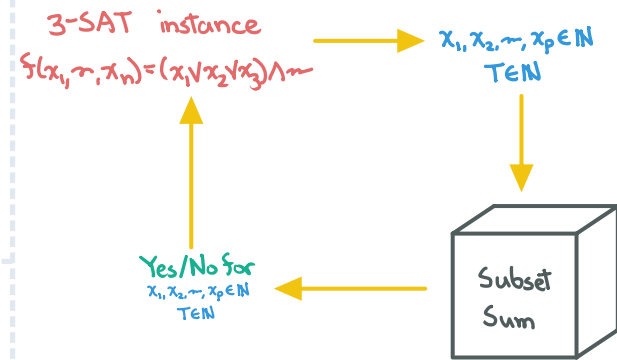
$$x_j \text{th digit of } D = 1$$

Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: Find a subset of  $x_i$ 's that sum to  $T$   
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10  
 $T=52$



1. Recast 3SAT as "subset" problem  
 $X = \text{literals } \{x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n\}$

clause  $(x_1 \vee \bar{x}_2 \vee x_3) = \text{set } \{x_1, \bar{x}_2, x_3\} \subset X$

Goal: Find SCX st.

(a) for each clause  $C_i$ ,  $C_i \cap S \neq \emptyset$

(b) for each  $x_j$ ,  $S$  contains <sup>(exactly)</sup> 1 of  $\{x_j, \bar{x}_j\}$

2. Start making #'s:

• we use mtn digits base 4

• for the literal  $x_j$ :

$$A_j = \underbrace{0 \ 1 \ 0 \ 1}_{m \text{ slots for each clause}} \underbrace{0 \ 0 \ 1 \ 0}_{n \text{ slots for each variable}}$$

(models)  $x_j$

• for each clause  $C_i$ , set  $C_i$ th digit to 1 if  $x_j \in C_i$  (not  $\bar{x}_j$ ), else 0

• set " $x_j$ th digit" to 1, else 0

Similar for the literal  $\bar{x}_j$

Key idea: numbers are bit-strings too!

• use many digits • each digit encodes 1 constraint

Literals

Long #'s

$$X = \{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\} \leftrightarrow \{A_1, \bar{A}_1, \dots, A_n, \bar{A}_n\}$$

Set SCX

$$D = \sum_{x_j \in S} A_j + \sum_{\bar{x}_j \in S} \bar{A}_j$$

$$\forall \text{ constraint } C_i, C_i \cap S \neq \emptyset \leftrightarrow \forall \text{ constraint } C_i, C_i \text{th digit of } D \geq 1$$

$$\forall \text{ variables } x_j, |\{x_j, \bar{x}_j\} \cap S| = 1 \leftrightarrow \forall \text{ variables } x_j, x_j \text{th digit of } D = 1$$

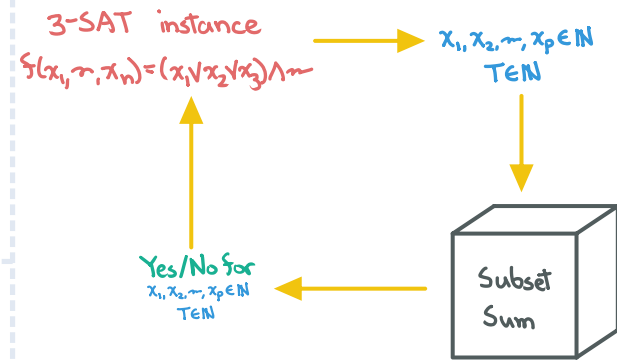
want to replace "≥" w/ "="

Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: Find a subset of  $x_i$ 's that sum to T  
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10  
T=52



1. Recast 3SAT as "subset" problem  
X = literals  $\{x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n\}$

clause  $(x_1 \vee \bar{x}_2 \vee x_3) = \text{set } \{x_1, \bar{x}_2, x_3\} \subset X$

Goal: Find SCX st.

(a) for each clause  $C_i$ ,  $C_i \cap S \neq \emptyset$

(b) for each  $x_j$ , S contains <sup>(exactly)</sup> 1 of  $\{x_j, \bar{x}_j\}$

2. Start making #'s:

• we use mtn digits base 4

• for the literal  $x_j$ :

$$A_j = \underbrace{0 \ 1 \ 0 \ 1}_{m \text{ slots for each clause}} \underbrace{0 \ 0 \ 1 \ 0}_{n \text{ slots for each variable}}$$

(models  $x_j$ )

• for each clause  $C_i$ , set  $C_i$ th digit to 1 if  $x_j \in C_i$  (not  $\bar{x}_j$ ), else 0

• set " $x_j$ th digit" to 1, else 0

Similar for the literal  $\bar{x}_j$

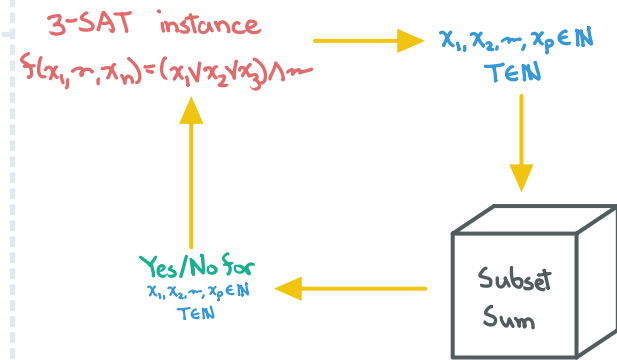
$\forall \text{ constraint } C_i, \leftrightarrow \forall \text{ constraint } C_i, C_i \cap S \neq \emptyset$

## Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: Find a subset of  $x_i$ 's that sum to  $T$   
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10  
 $T=52$



1. Recast 3SAT as "subset" problem

$X = \text{literals } \{x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n\}$   
clause  $(x_1 \vee \bar{x}_2 \vee x_3) = \text{set } \{x_1, \bar{x}_2, x_3\} \subset X$

Goal: Find  $S \subset X$  st.

(a) for each clause  $C_i, C_i \cap S \neq \emptyset$

(b) for each  $x_j, S$  contains exactly 1 of  $\{x_j, \bar{x}_j\}$

2. Start making #'s:

• we use mtn digits base 4

• for the literal  $x_j$ :

$A_j = \underbrace{0 \ 1 \ 0 \ 1}_{m \text{ slots for each clause}} \underbrace{0 \ 0 \ 1 \ 0}_{n \text{ slots for each variable}}$   
(models)  $x_j$

• for each clause  $C_i$ , set  $C_i$ th digit to 1 if  $x_j \in C_i$  (not  $\bar{x}_j$ ), else 0

• set " $x_j$ th digit" to 1, else 0

Similar for the literal  $\bar{x}_j$

Easy fix: "free padding"

for constraint  $C_i$ , create two copies of:

$B_i = 00 \sim 0100 \sim 0$

$\uparrow$   
 $C_i$ th digit

lets you increase  $C_i$ th digit by 1 or 2



Key idea: numbers are bit-strings too!

• use many digits • each digit encodes 1 constraint

Literals

Long #'s

$X = \{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\} \leftrightarrow \{A_1, \bar{A}_1, \dots, A_n, \bar{A}_n\}$   
and  $\{B_1, B_1, \dots, B_m, B_m\}$

let  $T = \underbrace{33\dots3}_m \text{ clauses } \underbrace{11\dots1}_n \text{ variables}$

for subset of  $A_i$ 's,  $\bar{A}_i$ 's,  $B_i$ 's summing to  $D$ :

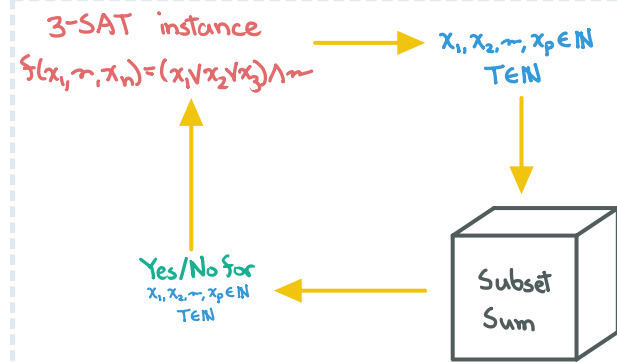
- $C_i$ th digit of  $D = 3$  only if  $D$  includes  $A_j/\bar{A}_j$  for some  $x_j/\bar{x}_j$  in  $C_i$
- $x_j$ th digit of  $D = 1 \iff$   
 $D$  includes (exactly) one of  $A_j/\bar{A}_j$

Subset Sum

Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$

Goal: find a subset of  $x_i$ 's that sum to  $T$   
(decision version: decide if such a subset sum exists)

e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10  
 $T = 52$



1. Recast 3SAT as "subset" problem  
 $X = \text{literals } \{x_1, \bar{x}_1, x_2, \bar{x}_2, \dots, x_n, \bar{x}_n\}$   
clause  $(x_1 \vee \bar{x}_2 \vee x_3) = \text{set } \{x_1, \bar{x}_2, x_3\} \subset X$

Goal: find  $S \subset X$  st.

- Ⓐ for each clause  $C_i$ ,  $C_i \cap S = \emptyset$
- Ⓑ for each  $x_j$ ,  $S$  contains <sup>(exactly)</sup> 1 of  $\{x_j, \bar{x}_j\}$

2. Start making #'s:

- we use mtrn digits base 4
- for the literal  $x_j$ :

$A_j = \underbrace{0 \ 1 \ 0 \ 1 \dots 1}_{m \text{ slots for each clause}} \underbrace{0 \ 0 \ 1 \ 1 \dots 1}_{n \text{ slots for each variable}}$   
(models  $x_j$ )

- for each clause  $C_i$ , set  $C_i$ th digit to 1 if  $x_j \in C_i$  (not  $\bar{x}_j$ ), else 0
  - set " $x_j$ th digit" to 1, else 0
- Similar for the literal  $\bar{x}_j$

Theorem: A polytime algo for subset-sum implies a polytime algo for SAT

Subset Sum  
 Input:  $x_1, x_2, \dots, x_n \in \mathbb{N}$   
 Goal: Find a subset of  $x_i$ 's that sum to  $T$   
 (decision version: decide if such a subset sum exists)  
 e.g. 14, 13, 17, 7, 18, 18, 10, 4, 11, 10  
 $T=52$

Literals  $X = \{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\} \leftrightarrow$  Long #'s  $\{A_1, \bar{A}_1, \dots, A_n, \bar{A}_n\}$   
 and  $\{B_1, \bar{B}_1, \dots, B_m, \bar{B}_m\}$

3-SAT instance

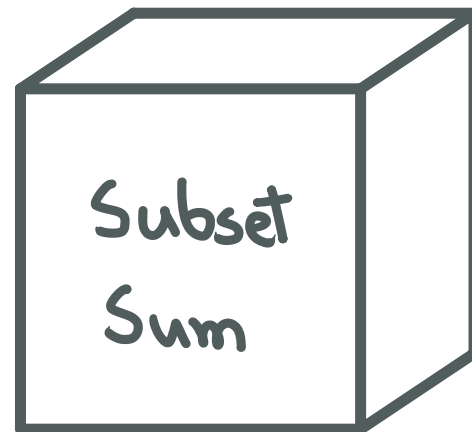
$$\mathcal{S}(x_1, \dots, x_n) = (x_1 \vee x_2 \vee x_3) \wedge \dots$$

$T = \underbrace{33 \sim 3}_{m \text{ clauses}} \underbrace{11 \sim 1}_{n \text{ variables}}$

$x_1, x_2, \dots, x_p \in \mathbb{N}$

$T \in \mathbb{N}$

Yes/No for  
 $x_1, x_2, \dots, x_p \in \mathbb{N}$   
 $T \in \mathbb{N}$



Key idea: numbers are bit-strings too!   
 — use many digits   
 — each digit encodes 1 constraint



