

- 1 Boolean functions
- 2 Representations
- 3 Satisfiability

- $f : \{false, true\}^n \rightarrow \{false, true\}$

# Boolean functions

- $f : \{false, true\}^n \rightarrow \{false, true\}$
- Example:
  - $XOR : \{false, true\}^2 \rightarrow \{false, true\}$

- $f : \{false, true\}^n \rightarrow \{false, true\}$
- Example:
  - $XOR : \{false, true\}^2 \rightarrow \{false, true\}$
  - $XOR(x_1, x_2) = x_1 \oplus x_2$
  - e.g.  $XOR(true, true) = true \oplus true = false$

- $f : \{false, true\}^n \rightarrow \{false, true\}$
- Example:
  - $XOR : \{false, true\}^2 \rightarrow \{false, true\}$
  - $XOR(x_1, x_2) = x_1 \oplus x_2$
  - e.g.  $XOR(true, true) = true \oplus true = false$
- In this course:  $false = 0$ ,  $true = 1$
- $XOR(1, 1) = 0$

Why are boolean functions interesting?

- Captures all functions on fixed input lengths
- Including basic operations within modern computers
- This will form the basis for understanding the reduction from any problem in **NP** to SAT (Cook's Theorem), giving us our basic **NP**-hard problem.

Ways of representing boolean functions:

- Truth tables
- Formulae
- Circuits

# Truth tables

- Explicitly enumerate all function values
- Example: XOR

$x_1$	$x_2$	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

- For each possible valuation of variables, list the function value

# Truth tables

- Explicitly enumerate all function values
- Example: XOR

$x_1$	$x_2$	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

- For each possible valuation of variables, list the function value
- “Compact” representation as a string: “0110”

# Truth tables

- Explicitly enumerate all function values
- Example: XOR

$x_1$	$x_2$	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

- For each possible valuation of variables, list the function value
- “Compact” representation as a string: “0110”
- For boolean function over  $n$  variables, string has length  $2^n$  (!)

- Variables  $x_1, x_2, \dots, x_n$  are fomulae.
- If  $f$  is a formula, then  $\neg(f)$  is a formula.
- If  $f_1$  and  $f_2$  are formulae, then  $(f_1) \wedge (f_2)$  is a formula.
- If  $f_1$  and  $f_2$  are formulae, then  $(f_1) \vee (f_2)$  is a formula.

# Formulae represent Boolean Functions

- “ $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$ ” represents the function *XOR*

# Formulae represent Boolean Functions

- “ $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$ ” represents the function *XOR*
- Several formulae can represent same function  
*XOR* is also represented by the formula  
“ $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ ”

# Formulae represent Boolean Functions

- “ $(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$ ” represents the function *XOR*
- Several formulae can represent same function  
*XOR* is also represented by the formula  
“ $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ ”
- Formula representation can be much more compact
- Never much less compact

# Construct a formula from a truth table

Construct a formula that is an “or” of the cases where the function evaluated to 1:

$x_1$	$x_2$	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

# Construct a formula from a truth table

Construct a formula that is an “or” of the cases where the function evaluated to 1:

$x_1$	$x_2$	$x_1 \oplus x_2$	$\neg x_1 \wedge x_2$	$x_1 \wedge \neg x_2$
0	0	0	0	0
0	1	1	1	0
1	0	1	0	1
1	1	0	0	0

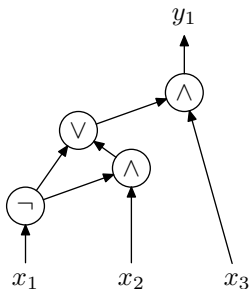
# Construct a formula from a truth table

Construct a formula that is an “or” of the cases where the function evaluated to 1:

$x_1$	$x_2$	$x_1 \oplus x_2$	$\neg x_1 \wedge x_2$	$x_1 \wedge \neg x_2$	$(\neg x_1 \wedge x_2) \vee (x_1 \wedge \neg x_2)$
0	0	0	0	0	0
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

Formulas can thus express any boolean function, using at most  $2^n$  clauses with  $n$  variables in each clause

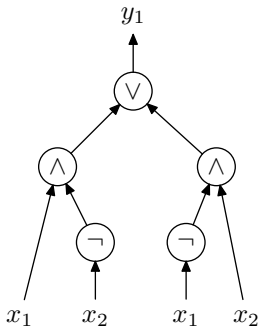
Example:



- A circuit  $C$  is a directed *acyclic* graph.
- Nodes in  $C$  are called *gates*.
- Types of gates:
  - *input* gates labeled  $x_1, \dots, x_n$  (in-degree 0)
  - Constant gates labeled 0 or 1 (in-degree 0)
  - $\wedge$  and  $\vee$  gates (in-degree 2)
  - $\neg$  and COPY gates (in-degree 1)
  - $m$  designated *output* gates
- The circuit  $C$  computes a boolean function  
 $C : \{0, 1\}^n \rightarrow \{0, 1\}^m$

# Formulae can be represented by a circuit

Example:



$$(x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$$

Formulas can thus be replaced by a circuit of comparable size.

# Are formulae and circuits equivalent?

- Not really, since formulae can have multiple output

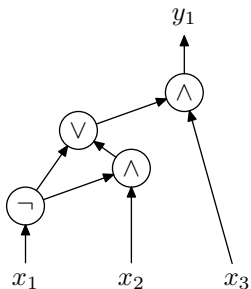
# Are formulae and circuits equivalent?

- Not really, since formulae can have multiple output
- But also not for single output circuits

# Are formulae and circuits equivalent?

- Not really, since formulae can have multiple output
- But also not for single output circuits
- We *can* write a formula for each gate in the circuit, but the formula may become much bigger than the circuit
- Circuits may reuse results of sub-computations

Example:



- For a function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ , let  $s_t = 2^n$  be the size (in bits) of its representation as a truth table

- For a function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ , let  $s_t = 2^n$  be the size (in bits) of its representation as a truth table
- Let  $s_f$  be the size (in bits) of its smallest representation as a formula. Then
  - $|s_f| \leq 10|s_t|^2$  (formulae are not much less compact than tables)
  - For all  $n$ , there is a function depending on all  $n$  variables so that  $|s_f| \leq 10(\log |s_t|)^2$  (for some function, formulae are *much* more compact)

# Tables – Formulae – Circuits

- For a function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ , let  $s_t = 2^n$  be the size (in bits) of its representation as a truth table
- Let  $s_f$  be the size (in bits) of its smallest representation as a formula. Then
  - $|s_f| \leq 10|s_t|^2$  (formulae are not much less compact than tables)
  - For all  $n$ , there is a function depending on all  $n$  variables so that  $|s_f| \leq 10(\log |s_t|)^2$  (for some function, formulae are *much* more compact)
- Let  $s_c$  be the size (in bits) of its smallest representation as a circuit. Then
  - $|s_c| \leq 10|s_f|^2$  (circuits are not much less compact than formulae)
  - Is it true that there, for all  $n$ , is a function so that circuits are *much* more compact (e.g.,  $|s_c| \leq 10(\log |s_f|)^2$ )?

# Tables – Formulae – Circuits

- For a function  $f : \{0, 1\}^n \rightarrow \{0, 1\}$ , let  $s_t = 2^n$  be the size (in bits) of its representation as a truth table
- Let  $s_f$  be the size (in bits) of its smallest representation as a formula. Then
  - $|s_f| \leq 10|s_t|^2$  (formulae are not much less compact than tables)
  - For all  $n$ , there is a function depending on all  $n$  variables so that  $|s_f| \leq 10(\log |s_t|)^2$  (for some function, formulae are *much* more compact)
- Let  $s_c$  be the size (in bits) of its smallest representation as a circuit. Then
  - $|s_c| \leq 10|s_f|^2$  (circuits are not much less compact than formulae)
  - Is it true that there, for all  $n$ , is a function so that circuits are *much* more compact (e.g.,  $|s_c| \leq 10(\log |s_f|)^2$ )? **This is open!**

- Example:  $f(x_1, x_2) = x_1 \oplus x_2$
- DNF: **D**isjunctive **N**ormal **F**orm  
Disjunction ( $\vee$ ) of *terms* (conjunction of literals)  
 $f(x_1, x_2) = (x_1 \wedge \neg x_2) \vee (\neg x_1 \wedge x_2)$
- CNF: **C**onjunctive **N**ormal **F**orm  
Conjunction ( $\wedge$ ) of *clauses* (disjunction of literals)  
 $f(x_1, x_2) = (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$

- SAT: Given a Boolean function in CNF representation, is there a way to assign truth values to the variables so that the function evaluates to true?
- SAT: Given a CNF, is it true that it does **not** represent the constant-0 function (there is at least one truth table entry with a 1 in it)
- Examples:
  - Input:  $(x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$   
Output: Yes
  - Input:  $(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$   
Output: No

- SAT is in **NP**.
- Cook's Theorem (1972): SAT is **NP**-hard.
- Hence, SAT is **NP**-complete.

# Reduction from CIRCUIT SAT to SAT

- Expanding each node did not work, as the size of the resulting formula could explode.

# Reduction from CIRCUIT SAT to SAT

- Expanding each node did not work, as the size of the resulting formula could explode.
- Introduce a new formula variable for each gate
- Add clauses forcing the new variable to have the correct value of the gate for all satisfying assignments.

# Reduction from CIRCUIT SAT to SAT

- Expanding each node did not work, as the size of the resulting formula could explode.
- Introduce a new formula variable for each gate
- Add clauses forcing the new variable to have the correct value of the gate for all satisfying assignments.
- $\neg$ -gate:  $h = \neg g$   
 $(h \vee g) \wedge (\neg h \vee \neg g)$

# Reduction from CIRCUIT SAT to SAT

- Expanding each node did not work, as the size of the resulting formula could explode.
- Introduce a new formula variable for each gate
- Add clauses forcing the new variable to have the correct value of the gate for all satisfying assignments.
- $\neg$ -gate:  $h = \neg g$   
 $(h \vee g) \wedge (\neg h \vee \neg g)$
- $\vee$ -gate:  $h = g_1 \vee g_2$   
 $(\neg h \vee g_1 \vee g_2) \wedge (h \vee \neg g_1) \wedge (h \vee \neg g_2)$

# Reduction from CIRCUIT SAT to SAT

- Expanding each node did not work, as the size of the resulting formula could explode.
- Introduce a new formula variable for each gate
- Add clauses forcing the new variable to have the correct value of the gate for all satisfying assignments.
- $\neg$ -gate:  $h = \neg g$   
 $(h \vee g) \wedge (\neg h \vee \neg g)$
- $\vee$ -gate:  $h = g_1 \vee g_2$   
 $(\neg h \vee g_1 \vee g_2) \wedge (h \vee \neg g_1) \wedge (h \vee \neg g_2)$
- $\wedge$ -gate:  $h = g_1 \wedge g_2$   
 $(h \vee \neg g_1 \vee \neg g_2) \wedge (\neg h \vee g_1) \wedge (\neg h \vee g_2)$