# VE527

## Computer-Aided Design of Integrated Circuits

Satisfiability (SAT)

# Satisfiability

- Called **SAT** for short
  - Given an appropriate representation of function $F(x_1, x_2, \ldots, x_n)$, find an assignment of the variables $(a_1, a_2, \ldots, a_n)$ so that $F(a_1, a_2, \ldots, a_n) = 1$.
  - Note: could have many satisfying solution; **any one** is fine.
  - However, if there are no satisfying assignments at all, prove it and return this info.
    - We call this **unSAT**.
- Something you can do with BDDs, can do **easier** with SAT.
  - SAT is aimed at scenarios where you just need **one satisfying assignment**…
  - … or prove that there is **no** such satisfying assignment.

# Example: Network Repair

- We want to find $(d_0, d_1, d_2, d_3)$ so that $(\forall ab\ z)(d_0, d_1, d_2, d_3) = 1$
- To repair the network, we only need **one satisfying assignment** for $(d_0, d_1, d_2, d_3)$.
- If **unSAT**, the network repair is impossible!

# Standard SAT Form: CNF

- **Conjunctive Normal Form (CNF)**
  - It is just standard **Product-of-Sums** form.

$$\Phi = \underbrace{(a + c)}_{\text{clause}}(b + \underbrace{c}_{\text{positive literal}})(\bar{a} + \bar{b} + \underbrace{\bar{c}}_{\text{negative literal}})$$

- Terminology
  - Each sum is called a **clause**.
  - Each variable in true form is called a **positive literal**.
  - Each variable in complement form is called a **negative literal**.
- Why CNF is useful?
  - Need only determine that **one** clause evaluates to "0" to know whole formula = "0".
  - Of course, to satisfy the whole formula, you must make **all** clauses identically "1".

# Assignment to a CNF Formula

- An **assignment** gives values to **some**, **not necessarily all**, of variables $x_i$ in $(x_1, x_2, \ldots, x_n)$.
  - **Complete** assignment: assigns values to all variables.
  - **Partial** assignment: some, not all, variables have values.
- Given an assignment, we can evaluate **status** of the clauses.
- There are three status:
  - **Conflicting**: Clause = 0
  - **Satisfied**: Clause=1
  - **Unsolved**: Clause unknown
- Example: $a = 0, b = 1$, but $c$ and $d$ unassigned.

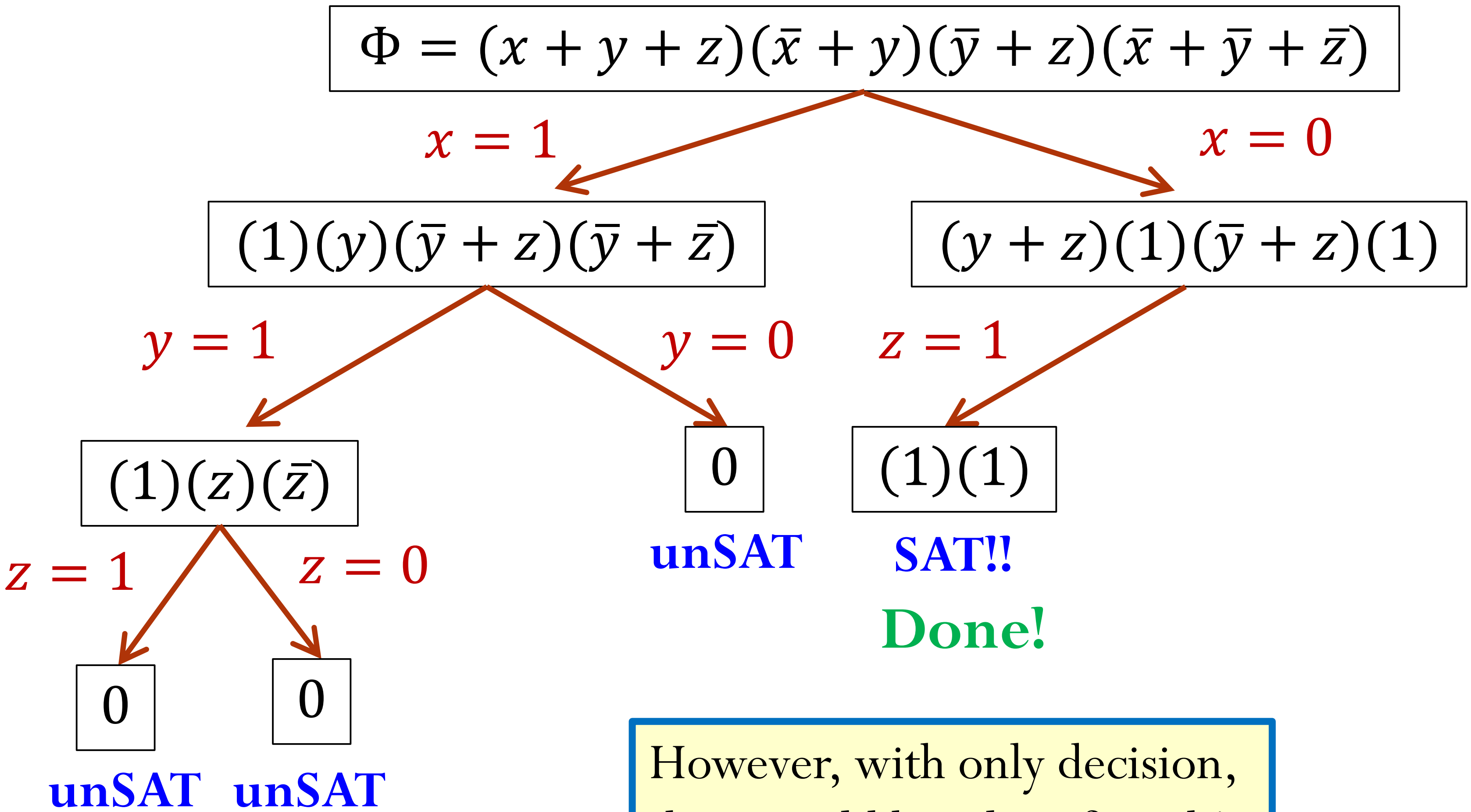
$$\Phi = \underbrace{(a + \bar{b})}_{\text{Conflicting}}\underbrace{(\bar{a} + b + \bar{c})}_{\text{Satisfied}}\underbrace{(a + c + d)}_{\text{Unsolved}}\underbrace{(\bar{a} + \bar{b} + \bar{c})}_{\text{Satisfied}}$$

# How to Solve SAT Problem?

- **Recursively!**

- Idea #1: **Decision**
  - Select a variable and **assign** its value; **simplify** CNF formula as far as you can.
  - Hope you can decide if it is SAT or unSAT, without any further work.
  - If you cannot, pick another variable.

# Decision: Example




However, with only decision, there could be a lot of work!

# How to Solve SAT Problem?

- Idea #2: **Deduction**
  - Look at the newly simplified clauses.
  - Based on **structure of clauses**, and **values of partial assignment**, we can **deduce** the values of some unassigned variables so that SAT is **possible**.
  - With new values deducted, simplify the CNF as far as you can.
  - Do deduction and simplification **iteratively** until nothing simplifies. At this time,
    - If you can decide SAT, great!
    - If you decide unSAT, you have to backtrack to change a decision.
    - If you cannot say SAT/unSAT, you have to make decision again.

# Deduction: Example

$$\Phi = (x + y + z)(\bar{x} + y)(\bar{y} + z)(\bar{x} + \bar{y} + \bar{z})$$

$x = 1$

$$(1)(y)(\bar{y} + z)(\bar{y} + \bar{z})$$

Deduction: $y = 1$

**Simplify**

$$(1)(z)(\bar{z})$$

Deduction: $z = 1$

**Simplify**

$$(1)(0)$$

**unSAT**

# BCP: Boolean Constraint Propagation

- To do "**deduction**", use **Boolean Constraint Propagation (BCP).**
  - Given a set of **fixed** variable assignments, you "**deduce**" about other necessary assignments by "**propagating constraints**".
    - What constraints? Each clause should be **satisfied**.
- Most famous BCP strategy is "**Unit Clause Rule**"
  - A clause is said to be "**unit**" if it has **exactly one** **unassigned literal**.
  - Unit clause has **exactly one** way to be satisfied, i.e., pick polarity that makes clause="1".
  - This choice is called an "**implication**".

# Example: Unit Clause Rule

$$\Phi = (a + c)(b + c)(\bar{a} + \bar{b} + \bar{c})(c + d + e)$$

- Assume $a = 1, b = 1$
- We can deduct that $c = 0$.

# BCP Example

Partial Assignment is $x_9 = 0, x_{10} = 0,$
$x_{11} = 0, x_{12} = 1, x_{13} = 1$

$$\Phi = \omega_1 \omega_2 \cdots \omega_9$$

| | | |
|---|---|---|
| $\omega_1 = \bar{x}_1 + x_2$ | | $\omega_1 = \bar{x}_1 + x_2$ |
| $\omega_2 = \bar{x}_1 + x_3 + x_9$ | **Simplify** | $\omega_2 = \bar{x}_1 + x_3$ |
| $\omega_3 = \bar{x}_2 + \bar{x}_3 + x_4$ | | $\omega_3 = \bar{x}_2 + \bar{x}_3 + x_4$ |
| $\omega_4 = \bar{x}_4 + x_5 + x_{10}$ | | $\omega_4 = \bar{x}_4 + x_5$ |
| $\omega_5 = \bar{x}_4 + x_6 + x_{11}$ | | $\omega_5 = \bar{x}_4 + x_6$ |
| $\omega_6 = \bar{x}_5 + \bar{x}_6$ | | $\omega_6 = \bar{x}_5 + \bar{x}_6$ |
| $\omega_7 = x_1 + x_7 + \bar{x}_{12}$ | | $\omega_7 = x_1 + x_7$ |
| $\omega_8 = x_1 + x_8$ | | $\omega_8 = x_1 + x_8$ |
| $\omega_9 = \bar{x}_7 + \bar{x}_8 + \bar{x}_{13}$ | | $\omega_9 = \bar{x}_7 + \bar{x}_8$ |

**No SAT**
**No BCP**
**Now what?**

# BCP Example (cont.)

- Next: Assign a variable to a value
  - Assign $x_1 = 1$

| | | |
|---|---|---|
| $\omega_1 = \bar{x}_1 + x_2$ | | $\omega_1 = x_2$ |
| $\omega_2 = \bar{x}_1 + x_3$ | | $\omega_2 = x_3$ |
| $\omega_3 = \bar{x}_2 + \bar{x}_3 + x_4$ | Simplify | $\omega_3 = \bar{x}_2 + \bar{x}_3 + x_4$ |
| $\omega_4 = \bar{x}_4 + x_5$ | | $\omega_4 = \bar{x}_4 + x_5$ |
| $\omega_5 = \bar{x}_4 + x_6$ | | $\omega_5 = \bar{x}_4 + x_6$ |
| $\omega_6 = \bar{x}_5 + \bar{x}_6$ | | $\omega_6 = \bar{x}_5 + \bar{x}_6$ |
| $\omega_7 = x_1 + x_7$ | | $\omega_7 = 1$ |
| $\omega_8 = x_1 + x_8$ | | $\omega_8 = 1$ |
| $\omega_9 = \bar{x}_7 + \bar{x}_8$ | | $\omega_9 = \bar{x}_7 + \bar{x}_8$ |

Implication

$x_2 = 1$

$x_3 = 1$

# BCP Example (cont.)

- Assign implied values
  - Assign $x_2 = 1, x_3 = 1$

| | | | |
|---|---|---|---|
| $\omega_1 = x_2$ | | $\omega_1 = 1$ | |
| $\omega_2 = x_3$ | | $\omega_2 = 1$ | **Implication** |
| $\omega_3 = \bar{x}_2 + \bar{x}_3 + x_4$ | **Simplify** $\Rightarrow$ | $\omega_3 = x_4$ | $\rightarrow$ $x_4 = 1$ |
| $\omega_4 = \bar{x}_4 + x_5$ | | $\omega_4 = \bar{x}_4 + x_5$ | |
| $\omega_5 = \bar{x}_4 + x_6$ | | $\omega_5 = \bar{x}_4 + x_6$ | |
| $\omega_6 = \bar{x}_5 + \bar{x}_6$ | | $\omega_6 = \bar{x}_5 + \bar{x}_6$ | |
| $\omega_7 = 1$ | | $\omega_7 = 1$ | |
| $\omega_8 = 1$ | | $\omega_8 = 1$ | |
| $\omega_9 = \bar{x}_7 + \bar{x}_8$ | | $\omega_9 = \bar{x}_7 + \bar{x}_8$ | |

# BCP Example (cont.)

- Assign implied values
  - Assign $x_4 = 1$

$\omega_1 = 1$

$\omega_2 = 1$

$\omega_3 = x_4$

$\omega_4 = \bar{x}_4 + x_5$

$\omega_5 = \bar{x}_4 + x_6$

$\omega_6 = \bar{x}_5 + \bar{x}_6$

$\omega_7 = 1$

$\omega_8 = 1$

$\omega_9 = \bar{x}_7 + \bar{x}_8$

**Simplify**

$\omega_1 = 1$

$\omega_2 = 1$

$\omega_3 = 1$

$$\left.\begin{array}{l}\omega_4 = x_5 \\ \omega_5 = x_6\end{array}\right\} \xrightarrow{\text{Implication}} \begin{array}{l}x_5 = 1 \\ x_6 = 1\end{array}$$

$\omega_6 = \bar{x}_5 + \bar{x}_6$

$\omega_7 = 1$

$\omega_8 = 1$

$\omega_9 = \bar{x}_7 + \bar{x}_8$

# BCP Example (cont.)

- Assign implied values
  - Assign $x_5 = 1, x_6 = 1$

$\omega_1 = 1$
$\omega_2 = 1$
$\omega_3 = 1$
$\omega_4 = x_5$
$\omega_5 = x_6$
$\omega_6 = \bar{x}_5 + \bar{x}_6$
$\omega_7 = 1$
$\omega_8 = 1$
$\omega_9 = \bar{x}_7 + \bar{x}_8$

**Simplify** $\Rightarrow$

$\omega_1 = 1$
$\omega_2 = 1$
$\omega_3 = 1$
$\omega_4 = 1$
$\omega_5 = 1$
$\omega_6 = 0$ $\rightarrow$ **Conflicting!**

**unSAT**

$\omega_7 = 1$
$\omega_8 = 1$
$\omega_9 = \bar{x}_7 + \bar{x}_8$

# BCP Example: Summary

- We start from partial assignment:
  $x_9 = 0, x_{10} = 0, x_{11} = 0,$
  $x_{12} = 1, x_{13} = 1$
- Next we assign $x_1 = 1$.
- After that, by BCP, we get implications:
  $x_2 = 1, x_3 = 1$
  $x_4 = 1$
  $x_5 = 1, x_6 = 1$
- Finally, we obtain a conflicting clause → unSAT

$$\Phi = \omega_1 \omega_2 \cdots \omega_9$$
$$\omega_1 = \bar{x}_1 + x_2$$
$$\omega_2 = \bar{x}_1 + x_3 + x_9$$
$$\omega_3 = \bar{x}_2 + \bar{x}_3 + x_4$$
$$\omega_4 = \bar{x}_4 + x_5 + x_{10}$$
$$\omega_5 = \bar{x}_4 + x_6 + x_{11}$$
$$\omega_6 = \bar{x}_5 + \bar{x}_6$$
$$\omega_7 = x_1 + x_7 + \bar{x}_{12}$$
$$\omega_8 = x_1 + x_8$$
$$\omega_9 = \bar{x}_7 + \bar{x}_8 + \bar{x}_{13}$$

# When Does BCP Finish?

- Three cases when BCP finishes:
  - **SAT**: Find a SAT assignment, all clauses resolve to "1". Return the assignment.
  - **Unresolved**: One or more clauses unresolved.
    - What's next? Pick another unassigned variable, and recurse more.
  - **unSAT**: Find conflict, one or more clauses evaluate to "0".
    - What's next? You need to **undo** one of the previous variable assignments, try again…

# DPLL Algorithm

- What we have covered is the basic idea behind the famous SAT-solving algorithm -- **Davis-Putnam-Logemann-Loveland (DPLL) Algorithm**.
  - Davis and Putnam published the basic recursive framework in 1960.
  - Davis, Logemann, and Loveland found smarter BCP, e.g., unit-clause rule, in 1962.
- Big ideas
  - A complete, systematic search of variable assignments.
  - Use CNF form for efficiency.
  - BCP makes search stop earlier, "**resolving**" more assignments without recursing more.

# SAT: Huge Progress Last ~20 Years

- DPLL is only the start…
- SAT has been subject of intense work and **great progress**.
  - Efficient data structures for clauses (so can search them fast).
  - Efficient variable selection heuristics (so can find lots of implications).
  - Efficient BCP mechanisms (because SAT spends MOST of its time here).
  - Learning mechanisms (find patterns of variables that NEVER lead to SAT, avoid them).
- Results: Good SAT codes that can do huge problems, fast.
  - 50,000 variables; 50,000,000 clauses

You see why not use BDD?

# SAT Solvers

- Many good solvers available online, open source.

- Examples
  - **MiniSAT**, from Niklas Eén, Niklas Sörensson in Sweden.
  - **CHAFF**, from Sharad Malik and students, Princeton University.
  - **GRASP**, from Joao Marques-Silva and Karem Sakallah, University of Michigan.
  - …and many others too.

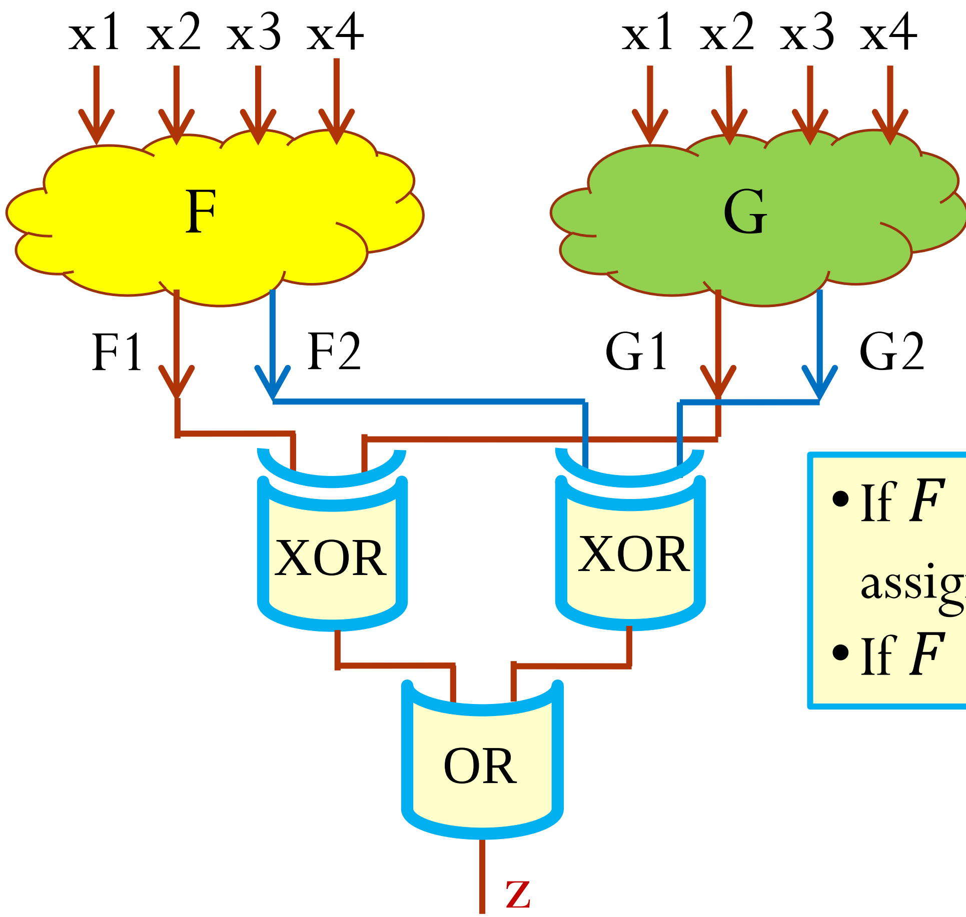
# BDD versus SAT Functionality

- BDD
  - Often work well for many problems.
  - But no guarantee always work.
  - Can build BDD to **represent function** $\Phi$.
    - Can do a big set of Boolean manipulations.
    - But sometimes cannot build BDD with reasonable computer resources (run out of memory SPACE)
  - Problem size **smaller** than SAT.

- SAT
  - Often work well for many problems.
  - But no guarantee always work.
  - Can **solve for SAT** (y/n) on function $\Phi$.
    - Does not support big set of operators.
    - But sometimes cannot find SAT with reasonable computer resources (run out of TIME doing search)
  - Problem size **larger** than BDD.

# Application of SAT in EDA

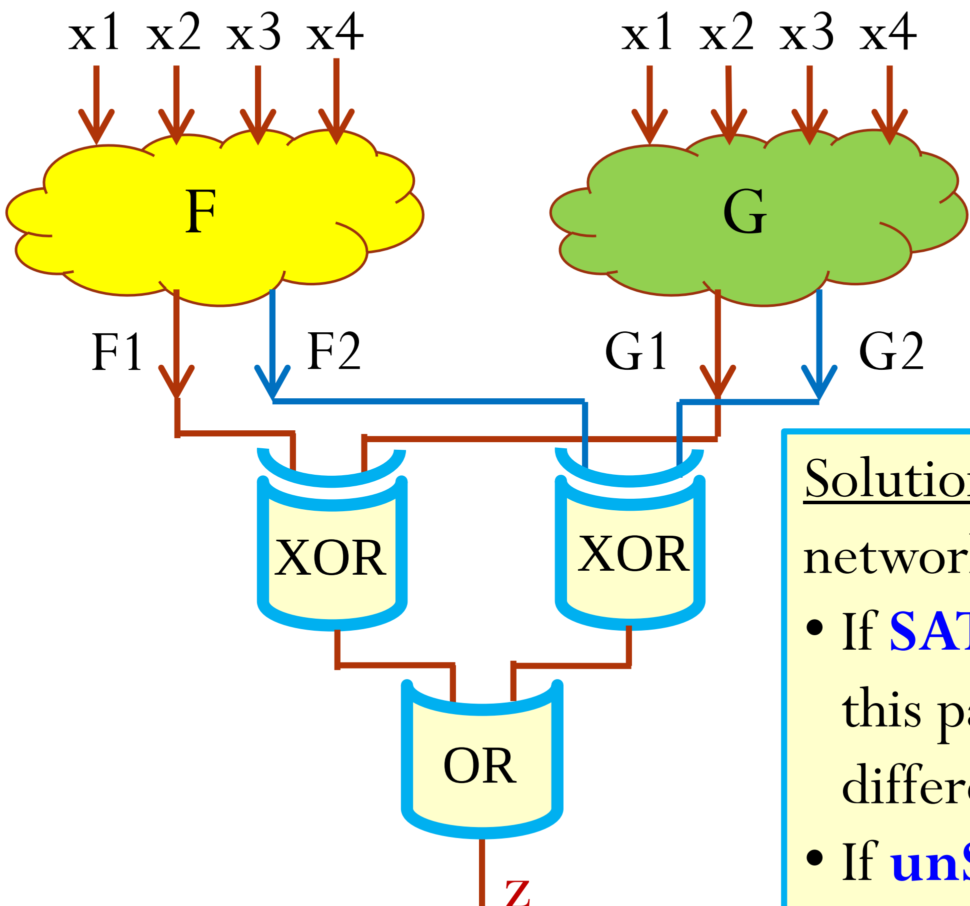
- Do these two logic networks implement the **same** Boolean function?




- If $F \neq G \rightarrow$ some input assignment lets $z = 1$: **SAT!**
- If $F = G \rightarrow z \equiv 0$: **unSAT**!

# Application of SAT in EDA

- Do these two logic networks implement the **same** Boolean function?

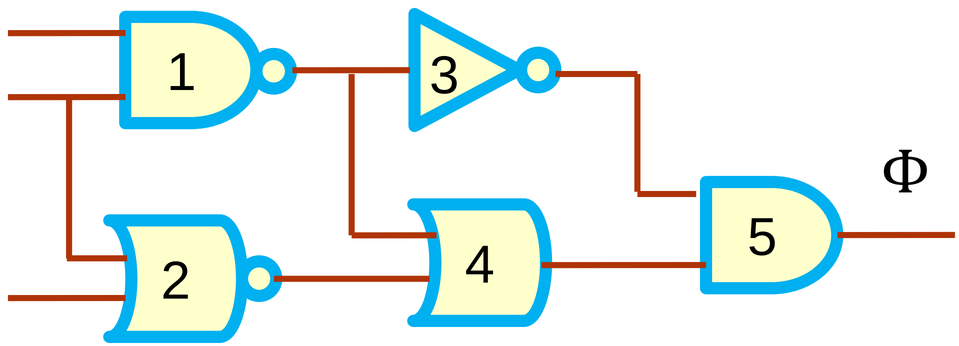



<u>Solution</u>: Do **SAT** on this new network

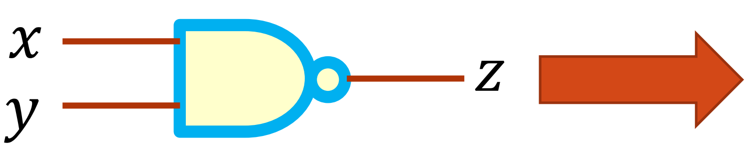
- If **SAT**: networks not same, and this pattern makes them give different outputs.
- If **unSAT**: yes, same!

# Related Question: Circuits → CNF

- How do we start with a gate-level description and get CNF?
  - Isn't this hard? No – it's really easy.




- <u>Idea</u>: build up CNF one gate at a time.
  - We build **gate consistency function** (or **gate satisfiability function**): $\Phi_z(x, y, z) = z\overline{\oplus}f(x, y)$

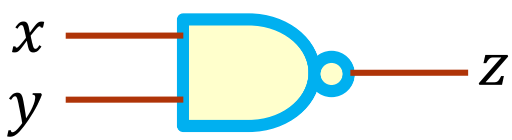



$$\Phi_z = z\overline{\oplus}\overline{xy}$$

$$\Phi_z = (x + z)(y + z)(\bar{x} + \bar{y} + \bar{z})$$

# Gate Consistency Function

- **Gate consistency function:** $\Phi_z(x, y, z) = z \overline{\oplus} f(x, y)$
  - It is "1" <u>**just**</u> for combinations of inputs and the output that are "consistent" with what gate actually does.




$$\Phi_z = (x + z)(y + z)(\bar{x} + \bar{y} + \bar{z})$$

**Consistent input**: $x = 0, y = 0, z = 1 \quad \Rightarrow \Phi_z = 1$

**Inconsistent input**: $x = 1, y = 1, z = 1 \quad \Rightarrow \Phi_z = 0$

# Rules for ALL Kinds of Basic Gates

$$z = x$$

$$(\bar{x} + z)(x + \bar{z})$$

$$z = \bar{x}$$

$$(x + z)(\bar{x} + \bar{z})$$

# Rules for ALL Kinds of Basic Gates

$z = \mathrm{NOR}(x_1, x_2, \ldots, x_n)$

$$\left[\prod_{i=1}^{n}(\bar{x}_i + \bar{z})\right]\left[\left(\sum_{i=1}^{n} x_i\right) + z\right]$$

$z = \mathrm{OR}(x_1, x_2, \ldots, x_n)$

$$\left[\prod_{i=1}^{n}(\bar{x}_i + z)\right]\left[\left(\sum_{i=1}^{n} x_i\right) + \bar{z}\right]$$

$z = \mathrm{NAND}(x_1, x_2, \ldots, x_n)$

$$\left[\prod_{i=1}^{n}(x_i + z)\right]\left[\left(\sum_{i=1}^{n} \bar{x}_i\right) + \bar{z}\right]$$

$z = \mathrm{AND}(x_1, x_2, \ldots, x_n)$

$$\left[\prod_{i=1}^{n}(x_i + \bar{z})\right]\left[\left(\sum_{i=1}^{n} \bar{x}_i\right) + z\right]$$

# Rules for ALL Kinds of Basic Gates

- XOR/XNOR gates are rather **unpleasant** for SAT solver.
  - They have rather large gate consistency functions.
  - Even small 2-input gates create a lot of terms.

$$z = x \oplus y$$

$$\Phi_z = z \overline{\oplus} (x \oplus y)$$
$$= (\bar{x} + \bar{y} + \bar{z})(x + y + \bar{z})$$
$$(x + \bar{y} + z)(\bar{x} + y + z)$$

$$z = x \overline{\oplus} y$$

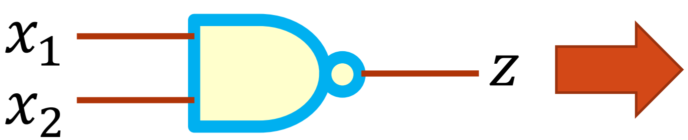
$$\Phi_z = z \overline{\oplus} (x \overline{\oplus} y)$$
$$= (x + y + z)(\bar{x} + \bar{y} + z)$$
$$(x + \bar{y} + \bar{z})(\bar{x} + y + \bar{z})$$

# Example: Apply the Rule

$$z = \mathrm{NAND}(x_1, x_2, \ldots, x_n)$$

$$\left[\prod_{i=1}^{n}(x_i + z)\right]\left[\left(\sum_{i=1}^{n}\bar{x}_i\right) + \bar{z}\right]$$

Example: $n = 2$

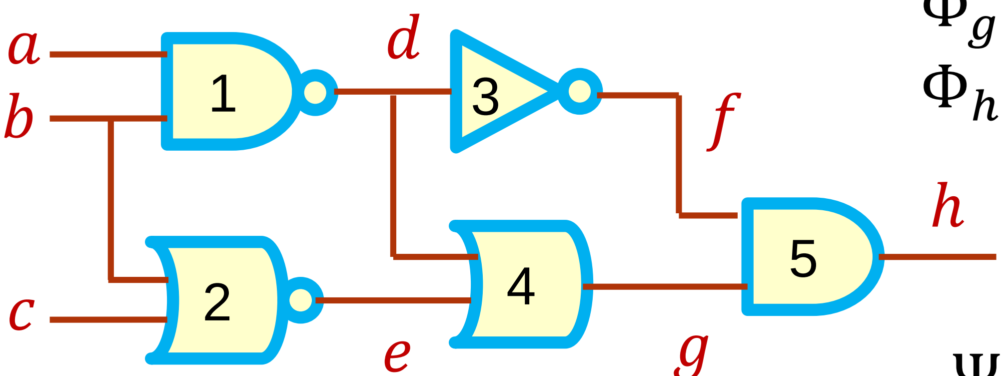



$$\Phi_z = (x_1 + z)(x_2 + z)(\bar{x}_1 + \bar{x}_2 + \bar{z})$$

# Circuits → CNF

- SAT CNF for network is simple:
  - Label each wire, build all gate consistency functions.
  - $\Psi = \boxed{(Output\ Var)} \cdot \prod_{k\ is\ gate\ output\ wire} \Phi_k$
    - Any pattern that satisfies the function also makes the gate network output=1.

$$\Phi_d = (a + d)(b + d)(\bar{a} + \bar{b} + \bar{d})$$
$$\Phi_e = (\bar{b} + \bar{e})(\bar{c} + \bar{e})(b + c + e)$$
$$\Phi_f = (\bar{d} + \bar{f})(d + f)$$
$$\Phi_g = (\bar{d} + g)(\bar{e} + g)(d + e + \bar{g})$$
$$\Phi_h = (f + \bar{h})(g + \bar{h})(\bar{f} + \bar{g} + h)$$




$$\Psi = h \cdot \Phi_d \cdot \Phi_e \cdot \Phi_f \cdot \Phi_g \cdot \Phi_h$$

# SAT
## Summary

- SAT has largely displaced BDDs for "just solve it" applications.
  - Reason is **scalability**: can do very large problems faster, more reliably.
  - Still, SAT, like BDDs, not guaranteed to find a solution in reasonable time or space.

- 50 years old, but still the big idea: **DPLL**
  - Many recent engineering advances make it amazingly fast.