First NP-complete problem—Circuit Satisfiability (problem definition)

- Boolean combinational circuit
  - Boolean combinational elements, wired together
  - Each element, inputs and outputs (binary)
  - Limit the number of outputs to 1.
  - Called *logic gates*: NOT gate, AND gate, OR gate.
  - *truth table*: giving the outputs for each setting of inputs
  - true assignment: a set of boolean inputs.
  - satisfying assignment: a true assignment causing the output to be 1.
  - A circuit is satisfiable if it has a satisfying assignment.

Circuit Satisfiability Problem: definition

- Circuit satisfying problem: given a boolean combinational circuit composed of AND, OR, and NOT, is it satisfiable?
- CIRCUIT-SAT={<C>: C is a satisfiable boolean circuit}

Circuit Satisfiability Problem: definition

- Circuit satisfying problem: given a boolean combinational circuit composed of AND, OR, and NOT, is it satisfiable?
- CIRCUIT-SAT={<C>: C is a satisfiable boolean circuit}
- Implication: in the area of computer-aided hardware optimization, if a subcircuit always produces 0, then the subcircuit can be replaced by a simpler subcircuit that omits all gates and just output a 0.

Two instances of circuit satisfiability problems

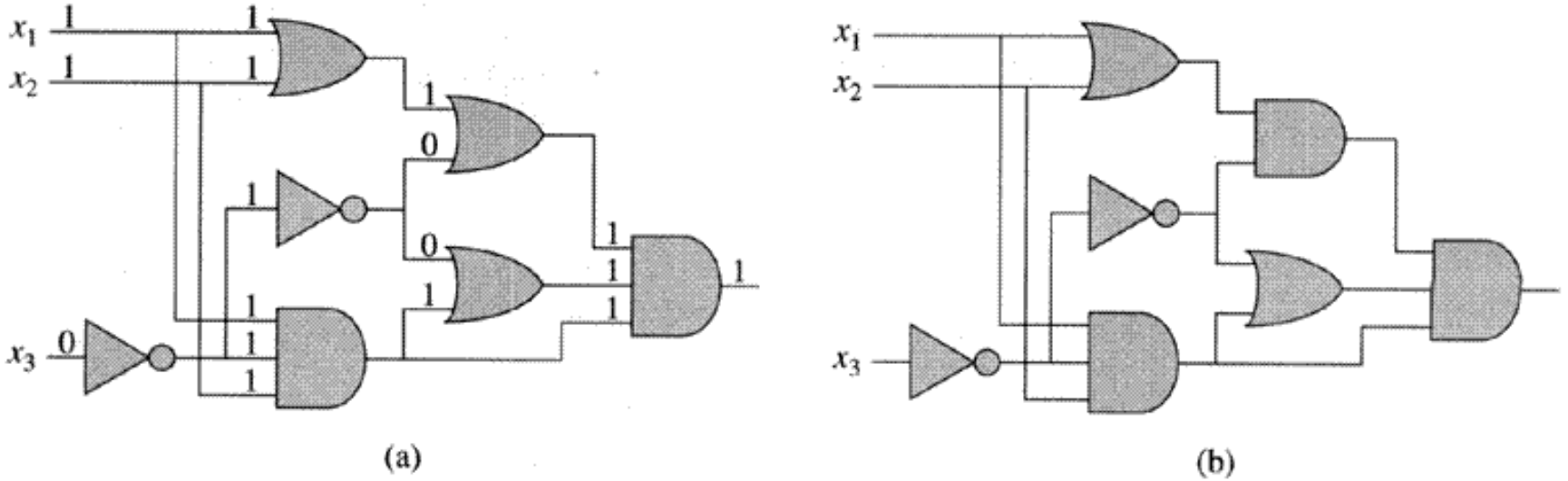


Figure 34.8 Two instances of the circuit-satisfiability problem. **(a)** The assignment $\langle x_1 = 1, x_2 = 1, x_3 = 0 \rangle$ to the inputs of this circuit causes the output of the circuit to be 1. The circuit is therefore satisfiable. **(b)** No assignment to the inputs of this circuit can cause the output of the circuit to be 1. The circuit is therefore unsatisfiable.

Solving circuit-satisfiability problem

- Intuitive solution:
  - for each possible assignment, check whether it generates 1.
  - suppose the number of inputs is k, then the total possible assignments are 2^k. So the running time is $\Omega(2^k)$. When the size of the problem is $\Theta(k)$, then the running time is not poly.

Circuit-satisfiability problem is NP-complete

- *Lemma :* CIRCUIT-SAT belongs to NP.

Circuit-satisfiability problem is NP-complete

- *Lemma :* CIRCUIT-SAT belongs to NP.
- Proof: CIRCUIT-SAT is poly-time verifiable.
  - Given (an encoding of) a CIRCUIT-SAT problem C and a certificate, which is an assignment of boolean values to (all) wires in C.
  - The algorithm is constructed as follows: just checks each gates and then the output wire of C:
    - If for every gate, the computed output value matches the value of the output wire given in the certificate and the output of the whole circuit is 1, then the algorithm outputs 1, otherwise 0.
    - The algorithm is executed in poly time (even linear time).

Circuit-satisfiability problem is NP-complete

- *Lemma :* CIRCUIT-SAT belongs to NP.
- Proof: CIRCUIT-SAT is poly-time verifiable.
  - Given (an encoding of) a CIRCUIT-SAT problem C and a certificate, which is an assignment of boolean values to (all) wires in C.
  - The algorithm is constructed as follows: just checks each gates and then the output wire of C:
    - If for every gate, the computed output value matches the value of the output wire given in the certificate and the output of the whole circuit is 1, then the algorithm outputs 1, otherwise 0.
    - The algorithm is executed in poly time (even linear time).
- An alternative certificate: a true assignment to the inputs.

Circuit-satisfiability problem is NP-complete (cont.)

- *Lemma :* CIRCUIT-SAT is NP-hard.

Circuit-satisfiability problem is NP-complete (cont.)

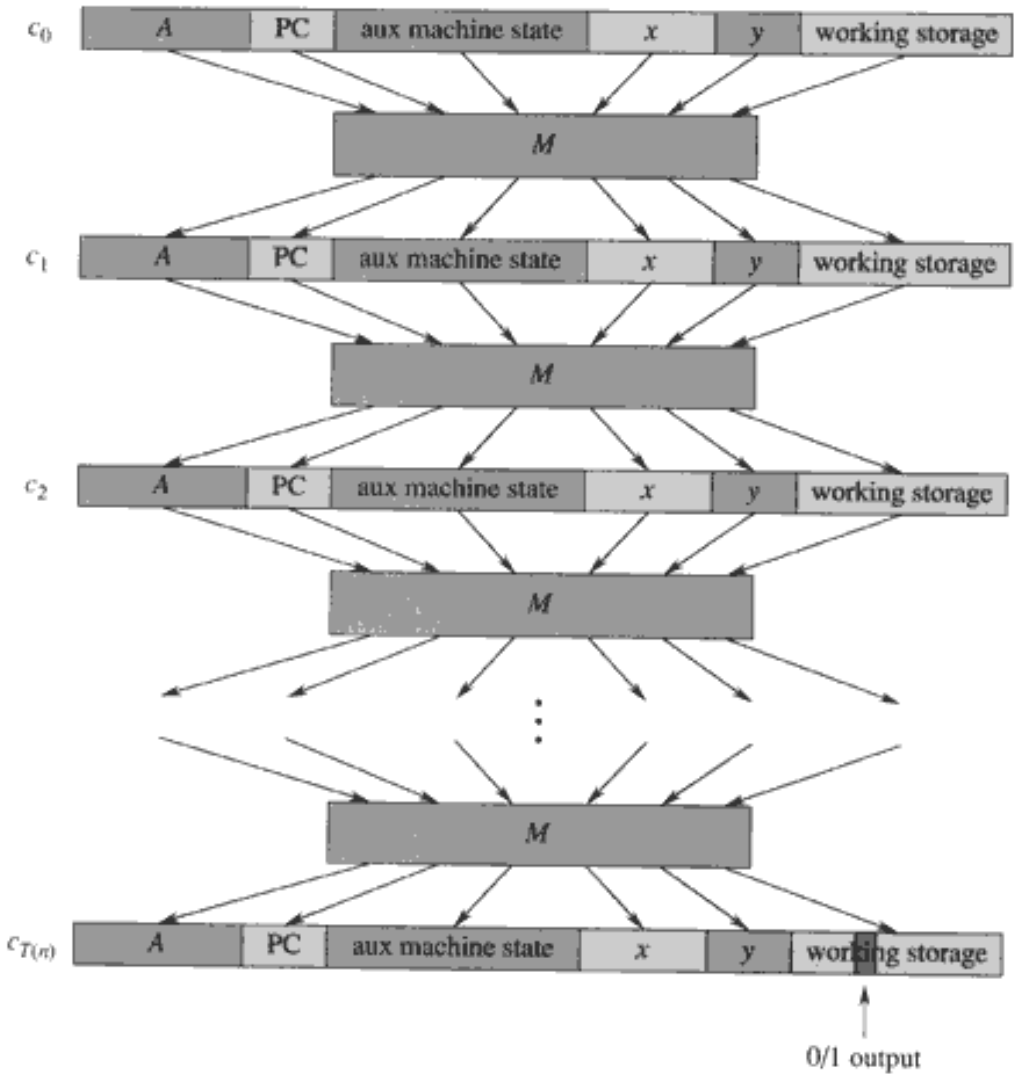
- *Lemma :* CIRCUIT-SAT is NP-hard.
- Proof: Suppose X is *any problem* in NP
  - construct a poly-time algorithm F maps every problem instance x in X to a circuit $C=f(x)$ such that the answer to x is YES if and only if $C \in$ CIRCUIT-SAT (is satisfiable).

Circuit-satisfiability problem is NP-hard (cont.)

- Since $X \in NP$, there is a poly-time algorithm A which verifies X.
- Suppose the input length is n and Let $T(n)$ denote the worst-case running time. Let k be the constant such that $T(n) = O(n^k)$ and the length of the certificate is $O(n^k)$.

Circuit-satisfiability problem is NP-hard (cont.)

- Idea is to represent the computation of A as a sequence of configurations, c_0, $c_1, \ldots, c_i, c_{i+1}, \ldots, c_{T(n)}$, each c_i can be broken into
  - (program for A, program counter PC, auxiliary machine state, input x, certificate y, working storage) and
  - c_i is mapped to c_{i+1} by the combinational circuit M implementing the computer hardware.
  - The output of A: 0 or 1– is written to some designated location in working storage. If the algorithm runs for at most $T(n)$ steps, the output appears as one bit in $c_{T(n)}$.
  - Note: $A(x,y)$=1 or 0.


c_0
A
PC
aux machine state
x
y
working storage
M
c_1
A
PC
aux machine state
x
y
working storage
M
c_2
A
PC
aux machine state
x
y
working storage
M
M
$c_{T(n)}$
A
PC
aux machine state
x
y
working storage
0/1 output

Circuit-satisfiability problem is NP-hard (cont.)

- The reduction algorithm F constructs a single combinational circuit C as follows:
  - Paste together all $T(n)$ copies of the circuit M.
  - The output of the ith circuit, which produces c_i, is directly fed into the input of the $(i+1)$st circuit.
  - All items in the initial configuration, except the bits corresponding to certificate y, are wired directly to their known values.
  - The bits corresponding to y are the inputs to C.
  - All the outputs to the circuit are ignored, except the one bit of $c_{T(n)}$ corresponding to the output of A.

Circuit-satisfiability problem is NP-hard (cont.)

- Two properties remain to be proven:

Circuit-satisfiability problem is NP-hard (cont.)

- Two properties remain to be proven:
  - F correctly constructs the reduction, i.e., C is satisfiable if and only if there exists a certificate y, such that $A(x,y)=1$.

Circuit-satisfiability problem is NP-hard (cont.)

- Two properties remain to be proven:
  - F correctly constructs the reduction, i.e., C is satisfiable if and only if there exists a certificate y, such that $A(x,y)=1$.

$\Leftarrow$Suppose there is a certificate y, such that $A(x,y)=1$. Then if we apply the bits of y to the inputs of C, the output of C is the bit of $A(x,y)$, that is $C(y)= A(x,y) =1$, so C is satisfiable.

Circuit-satisfiability problem is NP-hard (cont.)

- Two properties remain to be proven:
  - F correctly constructs the reduction, i.e., C is satisfiable if and only if there exists a certificate y, such that $A(x,y)=1$.

$\Leftarrow$Suppose there is a certificate y, such that $A(x,y)=1$. Then if we apply the bits of y to the inputs of C, the output of C is the bit of $A(x,y)$, that is $C(y)= A(x,y) =1$, so C is satisfiable.

$\Rightarrow$Suppose C is satisfiable, then there is a y such that $C(y)=1$. So, $A(x,y)=1$.

Circuit-satisfiability problem is NP-hard (cont.)

- Two properties remain to be proven:
  - F correctly constructs the reduction, i.e., C is satisfiable if and only if there exists a certificate y, such that $A(x,y)=1$.

$\Leftarrow$Suppose there is a certificate y, such that $A(x,y)=1$. Then if we apply the bits of y to the inputs of C, the output of C is the bit of $A(x,y)$, that is $C(y)= A(x,y) =1$, so C is satisfiable.

$\Rightarrow$Suppose C is satisfiable, then there is a y such that $C(y)=1$. So, $A(x,y)=1$.

  - F runs in poly time.

Circuit-satisfiability problem is NP-hard (cont.)

- F runs in poly time.
  - Poly space:
    - Size of x is n.
    - Size of A is constant, independent of x.
    - Size of y is $O(n^k)$.
    - Amount of working storage is poly in n since A runs at most $O(n^k)$.
    - M has size poly in length of configuration, which is poly in $O(n^k)$, and hence is poly in n.
    - C consists of at most $O(n^k)$ copies of M, and hence is poly in n.
    - Thus, the C has poly space.
  - The construction of C takes at most $O(n^k)$ steps and each step takes poly time, so F takes poly time to construct C from x.

CIRCUIT-SAT is NP-complete

- In summary
  - CIRCUIT-SAT belongs to NP, verifiable in poly time.
  - CIRCUIT-SAT is NP-hard, every NP problem can be reduced to CIRCUIT-SAT in poly time.
  - Thus CIRCUIT-SAT is NP-complete.

NP-completeness proof basis

- *Lemma:*
  - If X is a problem (class) such that $P' \leq_p X$ for some $P' \in$ NPC, then X is NP-hard. Moreover, if $X \in$ NP, then $X \in$ NPC.

NP-completeness proof basis

- *Lemma:*
  - If X is a problem (class) such that $P' \leq_p X$ for some $P' \in$ NPC, then X is NP-hard. Moreover, if $X \in$ NP, then $X \in$ NPC.
- Steps to prove X is NP-complete
  - Prove $X \in$ NP.
    - Given a certificate, the certificate can be verified in poly time.
  - Prove X is NP-hard.
    - Select a known NP-complete P'.
    - Describe a transformation function f that maps every instance x of P' into an instance $f(x)$ of X.
    - Prove f satisfies that the answer to $x \in$ P' is YES if and only if the answer to $f(x) \in$ X is YES for all instance $x \in$ P'.
    - Prove that the algorithm computing f runs in poly-time.

NPC proof –Formula Satisfiability (SAT)

- SAT definition
  - n boolean variables: $x_1, x_2, \ldots, x_n$.
  - M boolean connectives: any boolean function with one or two inputs and one output, such as $\wedge, \vee, \neg, \rightarrow, \leftrightarrow, \ldots$ and
  - Parentheses.
- A SAT ϕ is satisfiable if there exists a true assignment which causes ϕ to evaluate to 1.
- SAT=$\{< \phi >: \phi$ is a satifiable boolean formula$\}$.
- The historical honor of the first NP-complete problem ever shown.

SAT is NP-complete

- *Theorem:* SAT is NP-complete.
- Proof:
  - SAT belongs to NP.
    - Given a satisfying assignment, the verifying algorithm replaces each variable with its value and evaluates the formula, in poly time.
  - SAT is NP-hard (show CIRCUIT-SAT$\leq_p$ SAT).

SAT is NP-complete (cont.)

- CIRCUIT-SAT$\leq_p$ SAT, i.e., any instance of circuit satisfiability can be reduced in poly time to an instance of formula satisfiability.
- Intuitive induction:
  - Look at the gate that produces the circuit output.
  - Inductively express each of gate's inputs as formulas.
  - Formula for the circuit is then obtained by writing an expression that applies the gate's function to its input formulas.

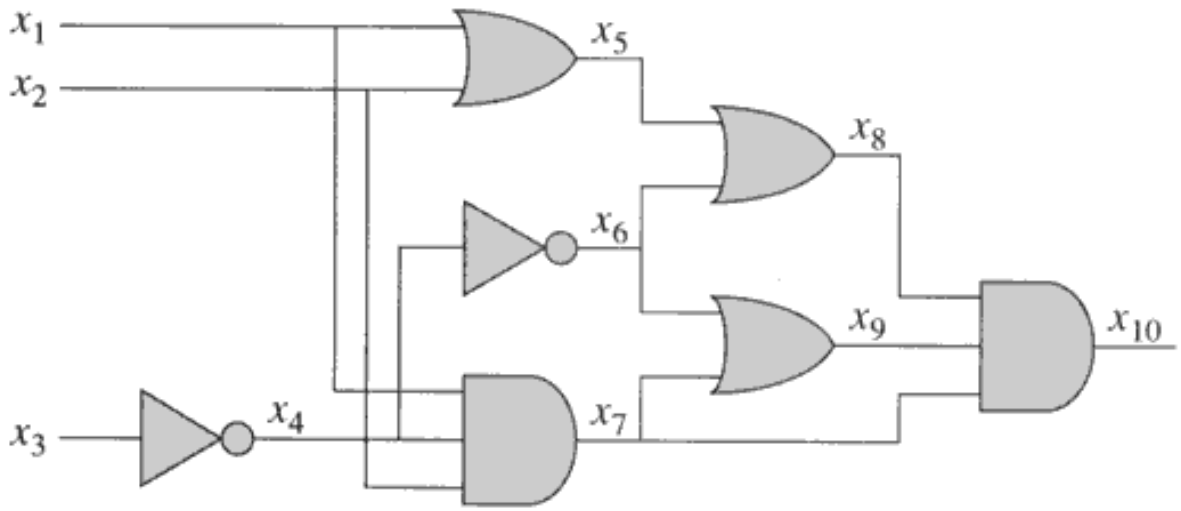
SAT is NP-complete (cont.)

- CIRCUIT-SAT$\leq_p$ SAT, i.e., any instance of circuit satisfiability can be reduced in poly time to an instance of formula satisfiability.
- Intuitive induction:
  - Look at the gate that produces the circuit output.
  - Inductively express each of gate's inputs as formulas.
  - Formula for the circuit is then obtained by writing an expression that applies the gate's function to its input formulas.

•Unfortunately, this is not a poly reduction

–Shared formula (the gate whose output is fed to 2 or more inputs of other gates) cause the size of generated formula to grow exponentially.

Example of reduction of CIRCUIT-SAT to SAT



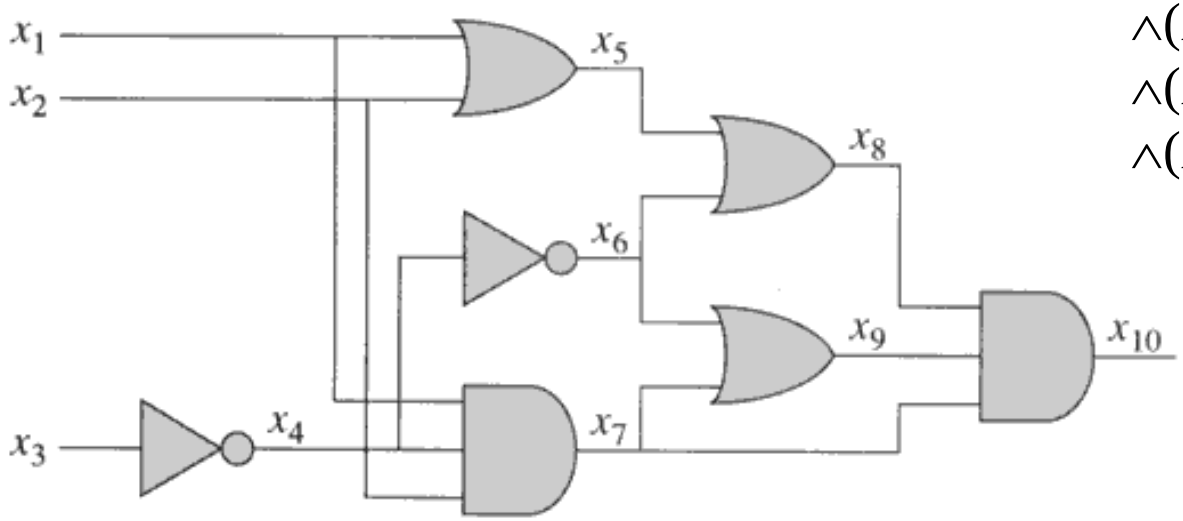
INCORRECT REDUCTION: $\phi = x_{10} = x_7 \wedge x_8 \wedge x_9 = (x_1 \wedge x_2 \wedge x_4) \wedge (x_5 \vee x_6) \wedge (x_6 \vee x_7)$
$= (x_1 \wedge x_2 \wedge x_4) \wedge ((x_1 \vee x_2) \vee \neg x_4) \wedge (\neg x_4 \vee (x_1 \wedge x_2 \wedge x_4)) = \ldots$

SAT is NP-complete (cont.)

- Correct reduction:
  - For every wire x_i of C, give a variable x_i in the formula.
  - Every gate can be expressed as $x_o \leftrightarrow (x_{i1} \theta\ x_{i2} \theta \ldots\ \theta\ x_{ij})$
  - The final formula ϕ is the AND of the circuit output variable and conjunction of all clauses describing the operation of each gate.

Example of reduction of CIRCUIT-SAT to SAT

$$\phi = x_{10} \wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)) \wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \wedge (x_6 \leftrightarrow \neg x_4) \wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \wedge (x_4 \leftrightarrow \neg x_3)$$



INCORRECT REDUCTION: $\phi = x_{10} = x_7 \wedge x_8 \wedge x_9 = (x_1 \wedge x_2 \wedge x_4) \wedge (x_5 \vee x_6) \wedge (x_6 \vee x_7)$
$= (x_1 \wedge x_2 \wedge x_4) \wedge ((x_1 \vee x_2) \vee \neg x_4) \wedge (\neg x_4 \vee (x_1 \wedge x_2 \wedge x_4)) = \ldots$

SAT is NP-complete (cont.)

- Correct reduction:
  - For every wire x_i of C, give a variable x_i in the formula.
  - Every gate can be expressed as $x_o \leftrightarrow (x_{i1} \theta\ x_{i2} \theta \ldots \theta\ x_{ij})$
  - The final formula ϕ is the AND of the circuit output variable and conjunction of all clauses describing the operation of each gate.
- Correctness of the reduction
  - Clearly the reduction can be done in poly time.
  - C is satisfiable if and only if ϕ is satisfiable.
    - If C is satisfiable, then there is a satisfying assignment. This means that each wire of C has a well-defined value and the output of C is 1. Thus the assignment of wire values to variables in ϕ makes each clause in ϕ evaluate to 1. So ϕ is 1.
    - The reverse proof can be done in the same way.

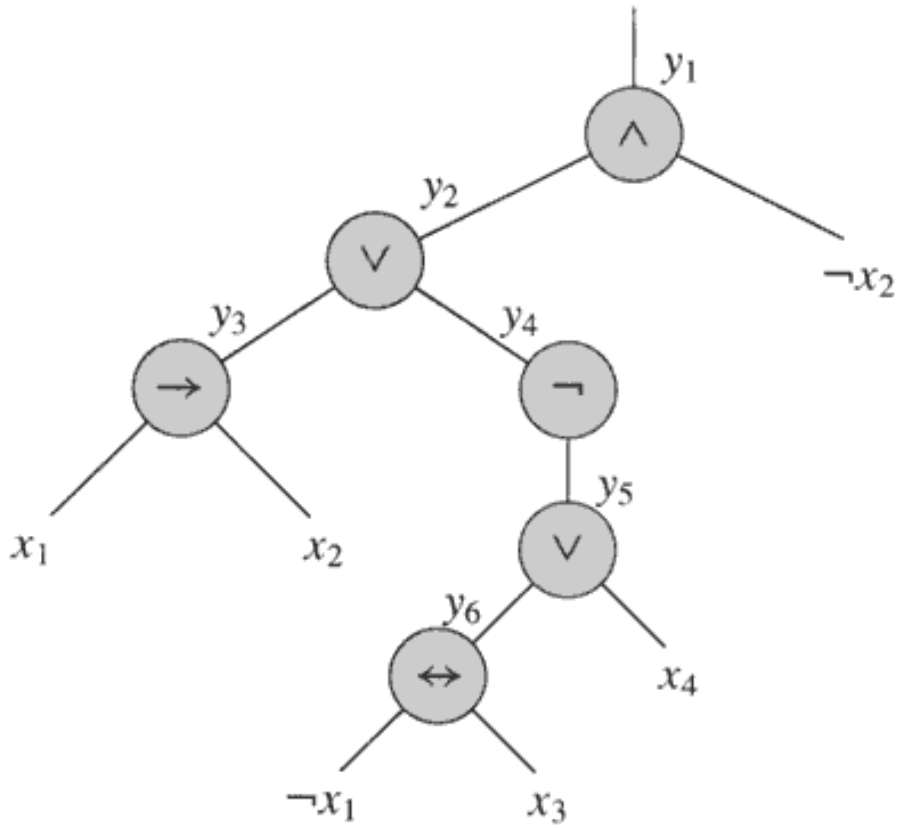
NPC Proof –3-CNF Satisfiability

- 3-CNF definition
  - A *literal* in a boolean formula is an occurrence of a variable or its negation.
  - CNF (Conjunctive Nornal Form) is a boolean formula expressed as AND of clauses, each of which is the OR of one or more literals.
  - 3-CNF is a CNF in which each clause has exactly 3 distinct literals (a literal and its negation are distinct)
- 3-CNF-SAT: whether a given 3-CNF is satiafiable?

3-CNF-SAT is NP-complete

- Proof: 3-CNF-SAT $\in$NP. Easy.
  - 3-CNF-SAT is NP-hard. (show SAT $\leq_p$3-CNF-SAT)
    - Suppose ϕ is any boolean formula, Construct a binary 'parse' tree, with literals as leaves and connectives as internal nodes.
    - Introduce a variable y_i for the output of each internal nodes.
    - Rewrite the formula to ϕ' as the AND of the root variable and a conjunction of clauses describing the operation of each node.
    - The result is that in ϕ', each clause has at most three literals.

Binary parse tree for $\phi=((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$



$$\begin{aligned}\phi' = {} & y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\ & \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \\ & \wedge (y_4 \leftrightarrow \neg y_5) \\ & \wedge (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \\ & \wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \\ & \wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))\end{aligned}$$

3-CNF-SAT is NP-complete(cont.)

- Proof: 3-CNF-SAT $\in$NP. Easy.
  - 3-CNF-SAT is NP-hard. (show SAT $\leq_p$3-CNF-SAT)
    - Change each clause into conjunctive normal form as follows:
      - Construct a true table, (small, at most 8 by 4)
      - Write the disjunctive normal form for all true-table items evaluating to 0
      - Using DeMorgan law to change to CNF.
    - The resulting ϕ" is in CNF but each clause has 3 or less literals.

Example of Converting a 3-literal clause to CNF format

y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	1

Disjunctive Normal Form:

$\phi_i' = (y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2)$
$\vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2)$

Conjunctive Normal Form:

$\phi_i'' = (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2)$
$\wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2)$

3-CNF-SAT is NP-complete(cont.)

- Proof: 3-CNF-SAT $\in$NP. Easy.
  - 3-CNF-SAT is NP-hard. (show SAT $\leq_p$3-CNF-SAT)
    - Change 1 or 2-literal clause into 3-literal clause as follows:
      - If a clause has one literal l, change it to $(l \vee p \vee q) \wedge (l \vee p \vee \neg q) \wedge (l \vee \neg p \vee q) \wedge (l \vee \neg p \vee \neg q)$.
      - If a clause has two literals $(l_1 \vee l_2)$, change it to $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$.

3-CNF is NP-complete

- ϕ and reduced 3-CNF are equivalent:
  - From ϕ to ϕ' , keep equivalence.
  - From ϕ' to ϕ'' , keep equivalence.
  - From ϕ'' to final 3-CNF, keep equivalence.
- Reduction is in poly time,
  - From ϕ to ϕ' , introduce at most 1 variable and 1 clause per connective in ϕ.
  - From ϕ' to ϕ'' , introduce at most 8 clauses for each clause in ϕ'.
  - From ϕ'' to final 3-CNF, introduce at most 4 clauses for each clause in ϕ''.