CNF Conversion and the Reduction from 3SAT to CLIQUE

CSCI3360 Introduction to Computational Complexity
Tutorial (Week 6)

Chak Fai Yung

Department of Computer Science and Engineering
The Chinese University of Hong Kong

Today's tutorial

1. **Proposition 3.3.1: Conversion to CNF.**
   Derive the clauses using implications, then prove the conversion.
2. **Section 4.3:** $3\mathrm{SAT} \leq_P \mathrm{CLIQUE}$**.**
   Encode a consistent choice of true literals as a clique.

As in Week 3: work through the construction, justify both directions, and bound the running time.

Literals, clauses, and CNF

- A **literal** is x_i or $\neg x_i$.
- A **clause** is an OR of literals.
- A **CNF formula** is an AND of clauses.

$$(x_1 \vee \neg x_2) \ \wedge \ (x_2 \vee x_3) \ \wedge \ (\neg x_1 \vee x_3).$$

Satisfaction

Every clause needs at least one true literal. All clauses must hold.

$F = (x_1 \wedge x_2) \vee \neg x_3$ is a Boolean formula, but its displayed form is not CNF.

Proposition 3.3.1

Polynomial-time conversion to CNF

Given $F(x_1, \ldots, x_n)$, construct a CNF formula $C(x_1, \ldots, x_n, z_1, \ldots, z_m)$ such that, for every original assignment α,

$$F(\alpha) = 1 \quad \Longleftrightarrow \quad \exists \beta \; C(\alpha, \beta) = 1.$$

The formula has size $O(|F|)$ and takes $O(|F|)$ time to construct.

The z_i are **auxiliary variables**. They record values of subformulas.

$$F(x) \equiv \exists z \, C(x, z).$$

The cost of direct distribution

$$(a \wedge b) \vee (c \wedge d) \equiv (a \vee c) \wedge (a \vee d) \wedge (b \vee c) \wedge (b \vee d).$$

For

$$F_t = (a_1 \wedge b_1) \vee \cdots \vee (a_t \wedge b_t),$$

straightforward distribution produces one clause for each choice of a_i or b_i from every pair: 2^t **clauses**.

A different representation

Auxiliary variables let us describe each connective separately.

The main tool: implication as an OR

Removing an implication

$$\boxed{(A \Rightarrow B) \equiv (\neg A \vee B)}$$

Both expressions are false **only when** $A = 1$ **and** $B = 0$.

A	B	$A \Rightarrow B$	$\neg A \vee B$
0	0	1	1
0	1	1	1
1	0	**0**	**0**
1	1	1	1

Example: $z \Rightarrow a$ becomes the clause $(\neg z \vee a)$.

Equivalence means two implications

To require z to equal a subformula G, enforce **both directions**:

$$z \leftrightarrow G \;\equiv\; (z \Rightarrow G) \wedge (G \Rightarrow z).$$

Apply $A \Rightarrow B \equiv \neg A \vee B$ to each direction.

Then simplify with double negation and De Morgan's laws:

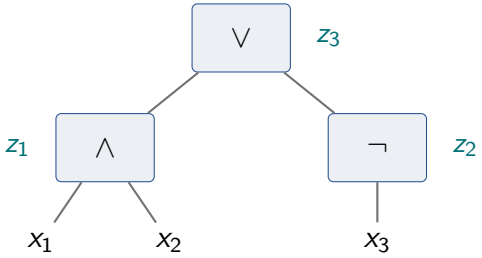
$$\begin{aligned} \neg\neg a &\equiv a, \\ \neg(a \wedge b) &\equiv \neg a \vee \neg b, \\ \neg(a \vee b) &\equiv \neg a \wedge \neg b. \end{aligned}$$

If an OR still contains an AND, use the distributive law locally.

One variable for each connective

$$F = (x_1 \wedge x_2) \vee \neg x_3.$$

$$z_1 \leftrightarrow (x_1 \wedge x_2),$$
$$z_2 \leftrightarrow \neg x_3,$$
$$z_3 \leftrightarrow (z_1 \vee z_2).$$



Use the original variables directly at the leaves. The root variable z_3 records the value of F.

The conversion algorithm

On input a formula F:

1. Traverse its syntax tree.
2. Give every connective node a fresh variable z_φ.
3. Add clauses enforcing the connective's relation to its children.
4. Add the unit clause (z_F), requiring the root to be true.
5. Return the conjunction of all clauses.

If F is just a variable x_i, return the unit clause (x_i).

Deriving the NOT clauses

Split $z \leftrightarrow \neg a$ into two implications:

$$(z \Rightarrow \neg a) \wedge (\neg a \Rightarrow z).$$

Use the implication rule in each direction:

$$\begin{aligned} z \Rightarrow \neg a &\equiv \neg z \vee \neg a, \\ \neg a \Rightarrow z &\equiv \neg(\neg a) \vee z \\ &\equiv a \vee z \qquad \text{(double negation)}. \end{aligned}$$

Conjoin the two clauses:

$$\boxed{(\neg z \vee \neg a) \wedge (z \vee a)}$$

Checking the NOT clauses

$$z \leftrightarrow \neg a \quad \equiv \quad (\neg z \vee \neg a) \wedge (z \vee a).$$

a	z	Clauses hold?	$z = \neg a$?
0	0	**no**	**no**
0	1	**yes**	**yes**
1	0	**yes**	**yes**
1	1	**no**	**no**

Each clause excludes one of the two incorrect assignments.

AND: the forward implication

For $z \leftrightarrow (a \wedge b)$, first consider $z \Rightarrow (a \wedge b)$.

$$
\begin{aligned}
z \Rightarrow (a \wedge b) &\equiv \neg z \vee (a \wedge b) && \text{(implication rule)} \\
&\equiv (\neg z \vee a) \wedge (\neg z \vee b) && \text{(distribution).}
\end{aligned}
$$

This gives **two clauses**:

$$\boxed{(\neg z \vee a) \qquad (\neg z \vee b)}$$

- If $z = 1$, the first clause forces $a = 1$ and the second forces $b = 1$.
- If $z = 0$, both clauses hold automatically.

AND: the reverse implication

The other direction is $(a \wedge b) \Rightarrow z$.

$$
\begin{aligned}
(a \wedge b) \Rightarrow z &\equiv \neg(a \wedge b) \vee z && \text{(implication rule)} \\
&\equiv (\neg a \vee \neg b) \vee z && \text{(De Morgan)} \\
&\equiv z \vee \neg a \vee \neg b.
\end{aligned}
$$

This clause forces $z = 1$ when $a = b = 1$.

Combining both directions gives the AND encoding:

$$\boxed{(\neg z \vee a) \wedge (\neg z \vee b) \wedge (z \vee \neg a \vee \neg b)}$$

Checking the AND constraint

a	b	Allowed z	A clause that excludes the wrong z
0	0	0	$\neg z \vee a$
0	1	0	$\neg z \vee a$
1	0	0	$\neg z \vee b$
1	1	1	$z \vee \neg a \vee \neg b$

Checkpoint: Suppose we keep only the first two clauses. What happens when $a = b = 1$?

Then $z = 0$ would also be allowed. The third clause is necessary.

OR: the forward implication

For $z \leftrightarrow (a \vee b)$, first consider $z \Rightarrow (a \vee b)$.

$$z \Rightarrow (a \vee b) \equiv \neg z \vee (a \vee b) \qquad \text{(implication rule)}$$
$$\equiv \neg z \vee a \vee b.$$

This is already **one clause**:

$$\boxed{(\neg z \vee a \vee b)}$$

- If $z = 1$, at least one of a, b must be true.
- If $a = b = 0$, the clause forces $z = 0$.

OR: the reverse implication

The other direction is $(a \vee b) \Rightarrow z$.

$$
\begin{aligned}
(a \vee b) \Rightarrow z &\equiv \neg(a \vee b) \vee z && \text{(implication rule)} \\
&\equiv (\neg a \wedge \neg b) \vee z && \text{(De Morgan)} \\
&\equiv (\neg a \vee z) \wedge (\neg b \vee z) && \text{(distribution).}
\end{aligned}
$$

Either true input therefore forces $z = 1$.

Combining both directions gives the OR encoding:

$$\boxed{(z \vee \neg a) \wedge (z \vee \neg b) \wedge (\neg z \vee a \vee b)}$$

The root clause

The gate clauses describe how to evaluate F. They allow the root to evaluate to either 0 or 1.

Requiring a satisfying assignment

Add the unit clause (z_F).

For $F = (x_1 \wedge x_2) \vee \neg x_3$, the root clause is (z_3).

$$C = \text{all gate clauses } \wedge \ z_3.$$

Without the root clause, even an assignment with $F(\alpha) = 0$ has a consistent extension recording its evaluation.

Worked example: the local clauses

$$F = (x_1 \wedge x_2) \vee \neg x_3.$$

$$\begin{aligned} A_1 &= (\neg z_1 \vee x_1) \wedge (\neg z_1 \vee x_2) \\ &\quad \wedge (z_1 \vee \neg x_1 \vee \neg x_2), \\ A_2 &= (\neg z_2 \vee \neg x_3) \wedge (z_2 \vee x_3), \\ A_3 &= (z_3 \vee \neg z_1) \wedge (z_3 \vee \neg z_2) \\ &\quad \wedge (\neg z_3 \vee z_1 \vee z_2). \end{aligned}$$

$$\boxed{C = A_1 \wedge A_2 \wedge A_3 \wedge z_3}$$

Three auxiliary variables and nine clauses in total.

Worked example: a satisfying assignment

Fix $(x_1, x_2, x_3) = (1, 0, 0)$.

$$F(1, 0, 0) = (1 \wedge 0) \vee \neg 0 = 1.$$

Evaluate the subformulas:

$$\begin{aligned} z_1 &= x_1 \wedge x_2 = 0, \\ z_2 &= \neg x_3 = 1, \\ z_3 &= z_1 \vee z_2 = 1. \end{aligned}$$

The extension $\beta = (0, 1, 1)$ satisfies every gate constraint and the root clause.

$$C\big((1, 0, 0), (0, 1, 1)\big) = 1.$$

Worked example: a falsifying assignment

Fix $(x_1, x_2, x_3) = (1, 0, 1)$, so $F(1, 0, 1) = 0$.

The local constraints force $z_1 = 0$ and $z_2 = 0$. The OR constraint then forces $z_3 = 0$.

- If $z_3 = 0$, the root clause (z_3) fails.
- If we try $z_3 = 1$, the clause $(\neg z_3 \vee z_1 \vee z_2)$ fails.

No auxiliary assignment can satisfy C for these original values.

$$\forall \beta \quad C\big((1, 0, 1), \beta\big) = 0.$$

Correctness: extending a satisfying assignment

Suppose $F(\alpha) = 1$.

1. For each subformula φ, set $z_\varphi = \varphi(\alpha)$.
2. Every connective now has the correct output for its inputs, so all local clauses hold.
3. Since $z_F = F(\alpha) = 1$, the root clause holds too.

$$F(\alpha) = 1 \quad \Longrightarrow \quad \exists \beta \; C(\alpha, \beta) = 1.$$

Correctness: recovering the original truth value

Suppose $C(\alpha, \beta) = 1$.

1. At a leaf, the value is the original variable's value under α.
2. Proceed upward. Each gate's clauses force its variable to equal the connective applied to its children.
3. Thus $z_\varphi = \varphi(\alpha)$ at every node, including $z_F = F(\alpha)$.
4. The root clause forces $z_F = 1$, hence $F(\alpha) = 1$.

$$\exists \beta \; C(\alpha, \beta) = 1 \quad \Longrightarrow \quad F(\alpha) = 1.$$

Size and construction time

Let g be the number of connective nodes in the syntax tree.

- At most g fresh variables.
- At most $3g + 1$ clauses, including the root clause.
- Each clause contains at most three literals.
- One traversal performs a constant amount of work at each node.

$$g = O(|F|) \quad \Longrightarrow \quad |C| = O(|F|), \qquad T(|F|) = O(|F|).$$

The conversion constructs a new formula. It does not search for a satisfying assignment.

Practice: encoding a negated AND

$$H = \neg(p \wedge q).$$

Introduce a for $p \wedge q$ and r for $\neg a$.

1. Split each gate equivalence into two implications, then derive its clauses using $A \Rightarrow B \equiv \neg A \vee B$.
2. Include the root clause.
3. How many clauses are there?
4. Can $(p, q) = (1, 1)$ extend to satisfy the result?

Discussion: What changes if the root clause is omitted?

Practice: the resulting CNF

$$C_H = (\neg a \vee p) \wedge (\neg a \vee q) \wedge (a \vee \neg p \vee \neg q)$$
$$\wedge (\neg r \vee \neg a) \wedge (r \vee a) \wedge r.$$

There are **six clauses**. For $p = q = 1$, the AND constraint forces $a = 1$ and the NOT constraint forces $r = 0$, contradicting the root clause.

Without the root clause, $a = 1, r = 0$ is a valid extension.

Auxiliary variables and existential quantification

The statement is

$$\boxed{F(x) \equiv \exists z\, C(x, z).}$$

- A true original assignment has a consistent extension.
- A false original assignment has no satisfying extension.
- An arbitrary choice of auxiliary values may violate the clauses.

For the worked example, $F(1, 0, 0) = 1$, but setting $z_1 = z_2 = z_3 = 0$ violates C.

Connection to exactly three literals per clause

Every clause produced above has **at most three literals**. Section 4.3 defines 3-CNF using **exactly three literal occurrences**.

The course permits repetition, so shorter clauses can be padded:

$$(\ell) \equiv (\ell \vee \ell \vee \ell), \qquad (\ell_1 \vee \ell_2) \equiv (\ell_1 \vee \ell_2 \vee \ell_2).$$

Padding preserves all assignments and keeps the size linear.

We can now use a 3-CNF formula as the input to the graph reduction.

A reduction connects two decision problems

Polynomial-time many-one reduction

Construct a function f in polynomial time such that

$$x \in A \iff f(x) \in B.$$

Our source instance is a formula; our target instance is a graph:

$$\langle \varphi \rangle \xrightarrow{f} \langle G_\varphi, k \rangle.$$

1. Describe the construction.
2. Prove both directions of the equivalence.
3. Bound the construction time in the input length.

The source problem: 3SAT

$$\varphi = C_1 \wedge \cdots \wedge C_m, \qquad C_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3}).$$

- A literal is x or $\neg x$.
- Every clause has **exactly three literal occurrences**.
- Repeated literals are allowed: $(x \vee y \vee y)$ is valid.
- Every clause needs at least one true literal.

The decision question

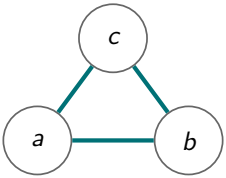
Does there exist a truth assignment satisfying all m clauses?

$$\mathrm{3SAT} = \{\langle\varphi\rangle : \varphi \text{ is a satisfiable 3-CNF formula}\}.$$

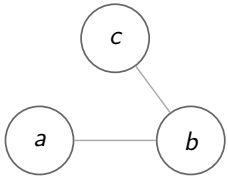
The target problem: CLIQUE

A clique is a set of vertices with an edge between **every pair**.

$$\text{CLIQUE} = \{\langle G, k\rangle : G \text{ has a clique of size at least } k\}.$$



a clique of size 3



one missing edge: no 3-clique

Theorem 4.3.1

$$3\text{SAT} \leq_{\text{P}} \text{CLIQUE}.$$

Running example: choose one true literal per clause

$$\varphi = C_1 \wedge C_2 \wedge C_3, \qquad \begin{aligned} C_1 &= (x_1 \vee \neg x_2 \vee x_3), \\ C_2 &= (\neg x_1 \vee x_2 \vee x_4), \\ C_3 &= (\neg x_3 \vee \neg x_4 \vee x_2). \end{aligned}$$

The assignment $(x_1, x_2, x_3, x_4) = (1, 1, 0, 0)$ satisfies φ.

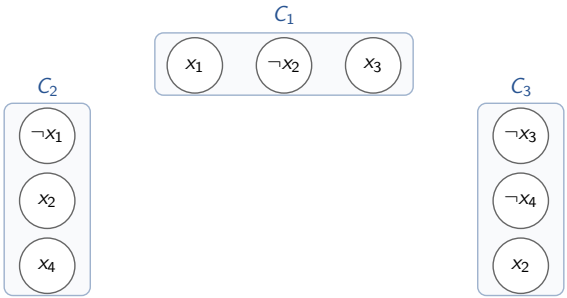
$$\underbrace{x_1}_{C_1} \qquad \underbrace{x_2}_{C_2} \qquad \underbrace{\neg x_3}_{C_3}$$

What the graph should encode

Choose one literal from each clause, with no contradictory choices.

Vertices represent literal occurrences

For each C_i, create three vertices $v_{i,1}, v_{i,2}, v_{i,3}$. Label $v_{i,r}$ by the literal $\ell_{i,r}$.



The two occurrences of x_2 are **two different vertices**.

Edges represent compatible choices across clauses

Add an edge between $v_{i,r}$ and $v_{j,s}$ exactly when

$$\boxed{i \neq j \quad \text{and} \quad \ell_{i,r}, \ell_{j,s} \text{ are not contradictory.}}$$

Same clause: no edge



These choices both come from C_1.

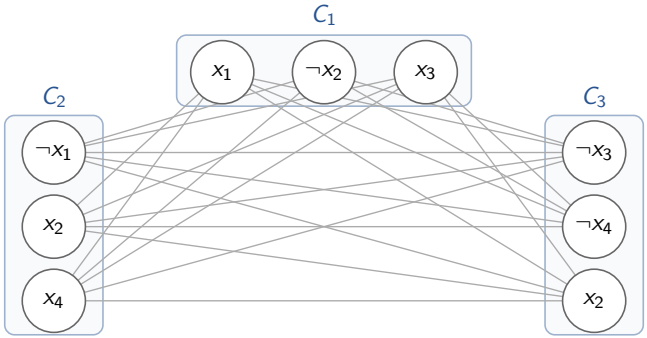
Opposite literals: no edge



Different clauses cannot make them compatible.

Different clauses may contain the **same** literal: that pair has an edge.

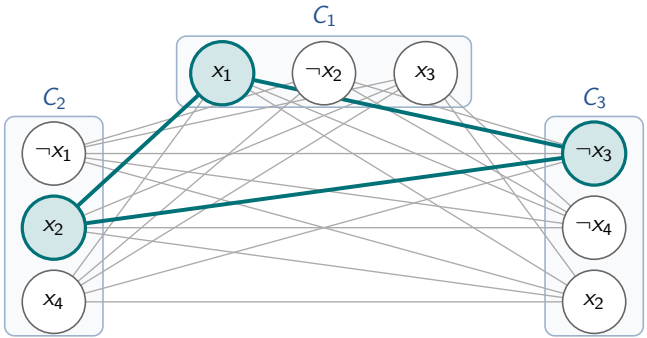
The complete graph for the course example



Nine vertices; edges only between compatible occurrences in different groups.

$$f(\langle\varphi\rangle) = \langle G_\varphi, 3\rangle.$$

The satisfying choices form a triangle



Selected: x_1 in C_1, x_2 in C_2, $\neg x_3$ in C_3.

Every pair is compatible and comes from different clauses: a 3-clique.

Forward direction: a satisfying assignment gives a clique

Suppose $\varphi = C_1 \wedge \cdots \wedge C_m$ is satisfiable.

1. From each clause, select one vertex whose literal is true.
2. This selects exactly m vertices from different clause groups.
3. Two selected literals cannot be x and $\neg x$: both are true under the same assignment.
4. The edge rule therefore connects every selected pair.

Conclusion

$$\varphi \text{ satisfiable} \quad \Longrightarrow \quad G_\varphi \text{ has an } m\text{-clique.}$$

Reverse direction: a clique must visit every clause

Suppose G_φ has a clique K of size at least m.

1. There are **no edges within a clause group**.
2. A clique therefore contains at most one vertex from each group.
3. There are m groups, so every clique has size at most m.

The threshold forces the desired choices

$|K| = m$, and K contains **exactly one vertex from every clause**.

Every pair in K is adjacent, so no selected literals contradict each other.

Reverse direction: consistent choices give an assignment

For the literals labeling vertices of K, assign

Selected literal	Variable value
x	$x = 1$
$\neg x$	$x = 0$

Assign any values to variables not represented in K.

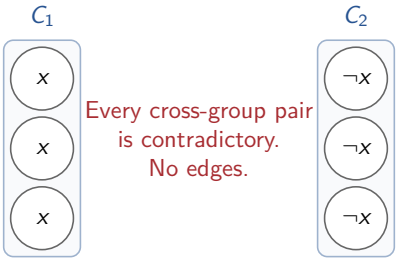
- The assignment is well-defined: K cannot select both x and $\neg x$.
- Every clause has a selected vertex, whose literal becomes true.

Conclusion

$$G_\varphi \text{ has an } m\text{-clique} \implies \varphi \text{ satisfiable.}$$

A no-instance remains a no-instance

$$\varphi = (x \vee x \vee x) \wedge (\neg x \vee \neg x \vee \neg x).$$



- φ requires both $x = 1$ and $x = 0$: it is unsatisfiable.
- The reduction outputs threshold $k = 2$.
- The graph has no edges, so it has no clique of size 2.

The graph has polynomial size and construction time

Let N be the encoded length of the 3-CNF formula with m clauses.

$$|V(G_\varphi)| = 3m, \qquad |E(G_\varphi)| \leq \binom{3m}{2} = O(m^2).$$

1. Create the three indexed vertices for each clause.
2. Examine every pair of vertices.
3. Compare clause indices, variable names, and polarities.
4. Output the explicit graph and $k = m$.

Cost measured in the input length

There are $O(m^2)$ pairs. Label comparisons and graph encoding take polynomial time in N; since $m \leq N$, the total cost is polynomial.

Practice: repeated labels and consistent choices

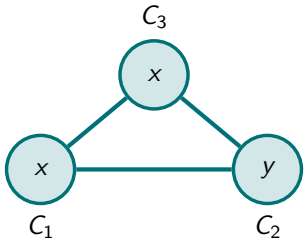
$$\psi = (x \vee y \vee y) \wedge (\neg x \vee y \vee y) \wedge (\neg y \vee x \vee x).$$

1. How many vertices does G_ψ have, and what is the threshold?
2. Is there an edge from x in clause 1 to $\neg x$ in clause 2?
3. Is there an edge between two occurrences of x in clause 3?
4. Choose three vertices forming a clique, and recover an assignment.

Use occurrence positions to identify vertices.

Practice answer: equal literals can be compatible

1. Three clauses give **nine vertices** and threshold $k = 3$.
2. x and $\neg x$ in different clauses: no edge.
3. Two occurrences in clause 3: no edge, even if both are labeled x.
4. Choose $v_{1,1}$, $v_{2,2}$, and $v_{3,2}$, labeled x, y, x.



Recover $x = 1, y = 1$; every clause is satisfied.

What the reduction lets us conclude

To decide $3\mathrm{SAT}$ using a CLIQUE algorithm:

1. Build $\langle G_\varphi, m \rangle$ in polynomial time.
2. Run the clique decision algorithm on that instance.
3. Return its answer.

Both correctness directions ensure the answers agree.

The consequence established here

$$\mathrm{CLIQUE} \in \mathrm{P} \quad \Longrightarrow \quad 3\mathrm{SAT} \in \mathrm{P}.$$

The notes establish the NP-completeness of $3\mathrm{SAT}$ later.

Takeaways: choices, compatibility, and both directions

1. **Vertices are choices.** One literal occurrence per vertex.
2. **Groups are clauses.** An m-clique selects one per clause.
3. **Edges are compatibility.** Different clauses, no opposite literals.
4. **Both proof directions matter.**

$$\varphi \text{ satisfiable} \iff G_\varphi \text{ has a clique of size } m.$$

5. **The construction is efficient.** $3m$ vertices and $O(m^2)$ pairs.

$$\mathrm{3SAT} \leq_{\mathrm{P}} \mathrm{CLIQUE} \qquad \text{and therefore} \qquad \mathrm{CLIQUE} \in \mathrm{P} \Rightarrow \mathrm{3SAT} \in \mathrm{P}.$$