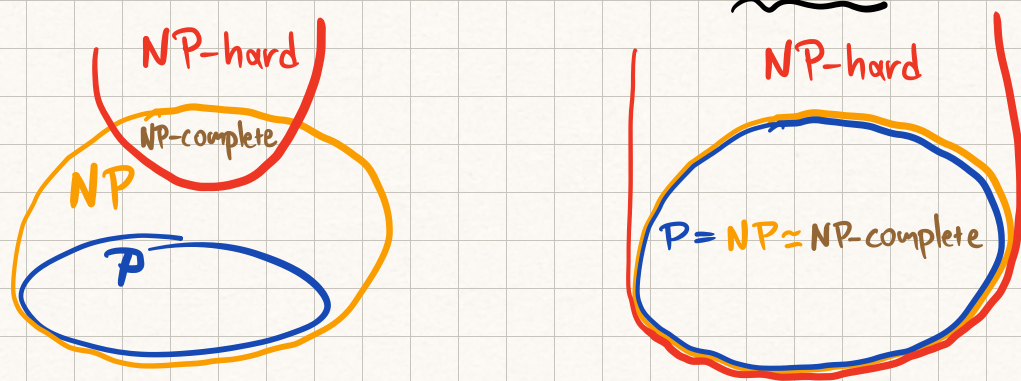
Lecture 6

- The P vs. NP question can be seen as a question about the power of nondeterminism in the Turing machine model. Recall that we resolved the same question on simpler computational models such as DFA vs. NFA and DPDA vs. NPDA.

- Interestingly, the above statement considers the NP definition using nondeterministic Turing machines. If we switch to the NP definition that uses a certificate, then we can rephrase the question as:

"Is recognizing the correct answer easier than coming up with an answer?"

$P \neq NP$ but if $P = NP$



*We will formally define NP-hard and NP-complete in this lecture.

? Let's entertain the idea of P=NP for a minute: If one can show that an NP-complete problem can be solved in polynomial time, then P=NP. The consequences would be tremendous! Mathematicians and engineers would be replaced by a program since the following language is NP-complete (so if P=NP we can construct proofs for any provable statement in poly-time)

$$\text{THEOREMS} = \{\langle \psi, 1^n \rangle \mid \psi \text{ has a formal proof of length } \leq n \text{ in axiomatic system } A\}$$

VLSI designers will be able to compute optimum circuits with minimum power requirements. We will be able to find the simplest formula/theory that explains any given data ⇒ Drug discovery and financial markets would be changed forever.

Interestingly, there would be no need for randomness. If P=NP, then there is no efficiency gain from using randomized algorithms

No privacy since every known efficient encryption scheme would be vulnerable to a reconstruction. ⇒ No hardness assumption (Public key crypto)
No One-Way functions (Symmetric key crypto)

NP-Completeness (Sipser, Chapter 7.4)

In the 1970s Cook and Levin discovered that some problems in NP had a structure that could be associated with all the problems in NP. These problems are called NP-complete and they have the property that if there is a polynomial time algorithm for any of these problems, then all problems in NP would be polynomially solvable. The first NP-complete problem we will see is called the satisfiability problem

Warm Up: Boolean variables can only take the values 1 (i.e., TRUE) or 0 (i.e., FALSE). Boolean operations AND, OR, and NOT are denoted as $\wedge$, $\vee$, and $\neg$. sometimes we write $\bar{x}$ instead of $\neg x$.

A Boolean formula is an expression involving Boolean variables and operations. for example $\phi = (\bar{x} \wedge y) \vee (x \wedge \bar{z})$.

A Boolean formula is satisfiable if there exists an assignment of 0s and 1s to the Boolean variables so that ϕ evaluates to 1.

The satisfiability problem is to test whether a given formula is satisfiable.

$$SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable Boolean formula}\}$$

Theorem 7.27: Language $SAT \in P$ if and only if $P = NP$

Polynomial Time Reducibility

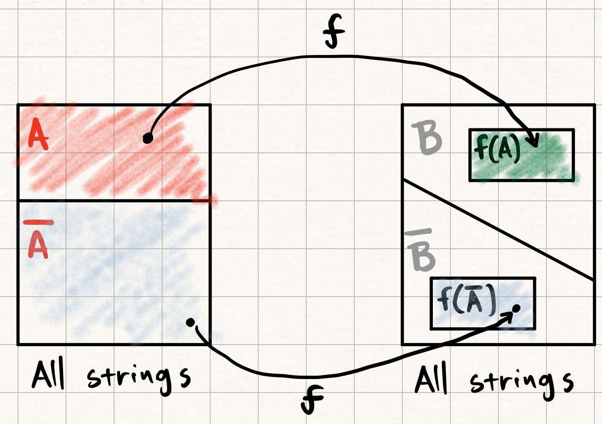
In Lecture 4 we saw that if problem A reduces to problem B, a solution to B can be used to solve A. In the following, we define a version of reducability that takes into account the efficiency of the reduction.

Definition 7.28: A function $f: \Sigma^* \to \Sigma^*$ is a polynomial time computable function if some polynomial time TM M exists that halts with just $f(w)$ on its tape, when started on any input w.

Definition 7.29: Language A is polynomial time (Karp) reducible to language B, denoted as $A \leq_p B$, if a polynomial time computable function $f: \Sigma^* \to \Sigma^*$ exists, where for every $w \in \Sigma^*$

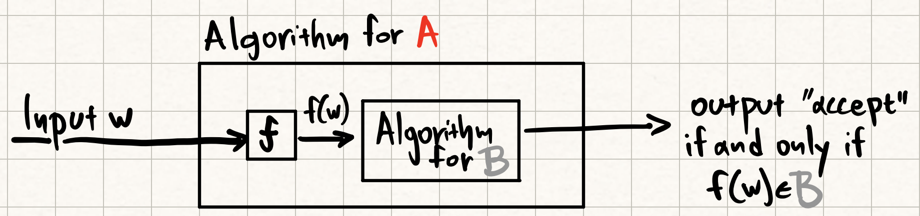
$$\boxed{w \in A \quad \text{if and only if} \quad f(w) \in B.}$$

Pictorially:

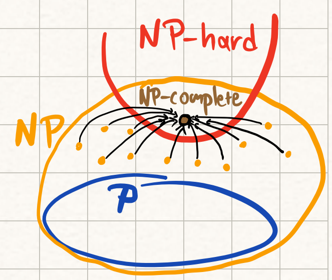


- The large rectangle represents all possible strings
- Notice that the lefthand side partitions all possible strings with respect to their membership (or not) to language A (or its compement $\overline{A} \triangleq \Sigma^* \setminus A$)
- Notice that f doesn't have to output all possible strings in B (which is why the green rectangle is a subset of B).

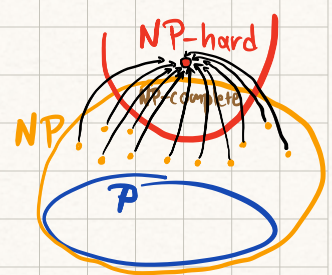
Transform a poly-time TM that decides B to a polytime TM that decides A:



Definition 7.34:
A language B is **NP-complete** if it satisfies two conditions
1. B is in NP, i.e., $B \in NP$
2. Every A in NP is polynomial time reducible to B



Definition:
A language B is **NP-hard** if $A \leq_p B$ for every $A \in NP$



Notice that a language doesn't have to be in NP to be NP-hard

Theorem: ① (Transitivity) If $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$
② If language B is NP-hard and $B \in P$, then $P = NP$
③ If language B is NP-complete and $B \in P$, then $P = NP$
④ If language B is NP-complete, then $B \in P$ if and only if $P = NP$

If $B \in P$ then $P = NP$ ⟶ ② says the same about NP-hard
and
$B \in P$ only if $P = NP$ ⟶ This doesn't hold for NP-hard problems. B might be harder than NP, i.e., $B \notin NP$

Remarks:

- An NP-hard problem is at least as hard as the hardest problems in NP.
- An NP-hard problem may even be undecidable (e.g., the undecidable "HALT" language is NP-hard).
- NP-complete is the class of decision problems in NP that are the hardest in NP.

Quiz 6.1: Which of the following statements is false?:
A) Language A can be both in P and NP
B) Language A can be both NP-complete and NP-hard
C) If language A is in NP and can be decided in poly-time, P=NP
D) There are no "harder" problems in NP than the NP-complete problems

Theorem 7.36: If B is NP-complete and $B \leq_p C$ and $C \in NP$, then C is NP-complete

The Cook-Levin Theorem: SAT is NP-complete

Proof Idea: To prove NP-completeness we first have to show that SAT is in NP.
A nondeterministic polynomial time machine can "guess" an assignment to a given formula ϕ and accept if the assignment satisfies ϕ.
For the second part, we need to show that every $L \in NP$ is polynomial time reducible to SAT.
Because we are focusing on an L that is in NP, we know from the NTM-based definition that there exists a NTM M′ that decides L.

The essence of the proof is to construct a Karp reduction that takes as an input a string w that may or may not be in L and translate it to a Boolean expression ϕ' such that:
- ϕ' is satisfiable if M′ accepts
- M′ accepts if ϕ' is satisfiable.

Thus, $w \in L \iff \phi' \in SAT$.
On a high-level the constructed Boolean formula ϕ' "simulates" the execution of the given NTM M′ that guarantees the membership $L \in NP$. ∎

*) The ingenuity of the Cook-Levin theorem is that
1. Can process ANY language $L \in NP$ without caring about the specifics of the problem, e.g., is it a graph theory problem? is it a number theory problem?
2. It is the very first NP-complete problem which means it didn't use Theorem 7.36 to reduce another NP-complete problem X to SAT

3SAT

A literal is a Boolean variable or a negated Boolean variable, e.g., x or $\bar{x}$.
A clause is a sequence of literals connected with ORs, e.g., $(x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4)$.
A Boolean formula is in conjuctive normal form, called CNF-formula, if it comprises several clauses connected with ANDs, e.g.,

$$(x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4) \wedge (x_3 \vee \bar{x}_5 \vee x_6) \wedge (x_3 \vee \bar{x}_6)$$

If all the clauses have at most k literals, then the form is called kCNF.
Language 3SAT is defined as:

$$3SAT = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable 3CNF-formula}\}.$$

(not in Sipser)

Theorem: $SAT \leq_p 3SAT$

Proof: (Sketch) We need to show that $3SAT \in NP$ and also present a polynomial time function that maps any CNF formula ϕ (valid input to test SAT membership) to a 3CNF formula ψ such that ψ is satisfiable if and only if ϕ is satisfiable.

A nondeterministic polynomial time machine can "guess" an assignment to a given 3CNF formula and accept if the assignment satisfies the formula.

Intuition for the mapping:

From 4CNF to 3CNF ⇝ Suppose we have $\phi = x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4$, then we can introduce a new variable z_1 and construct the following two clauses

$c_1: (x_1 \vee \bar{x}_2 \vee z_1)$ and $c_2: (\bar{x}_3 \vee x_4 \vee \bar{z}_1)$

If ϕ is satisfiable ⇒ there exists an assignment to z_1 that satisfies both c_1 and c_2
If c_1 and c_2 are satisfiable ⇒ there exists an assignment such that ϕ is satisfiable

Generalize: Suppose we have a clause of size k

$$\phi = (x_1 \vee \dots \vee x_{k-2} \vee \bar{x}_{k-1} \vee x_k)$$

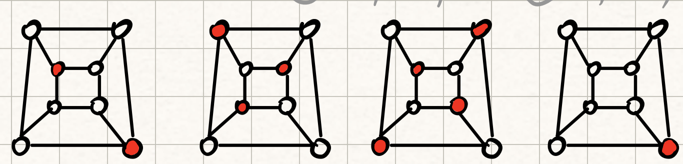
$c_1: (x_1 \vee \dots \vee x_{k-2} \vee \bar{z}_1)$ — k-1 literals ✗

$c_2: (\bar{x}_{k-1} \vee x_k \vee z_1)$ — 3 literals ✓

→ Apply this poly-time transformation until all clauses have at most 3 literals ∎

IND

An independent set of size k is a set of vertices S such that $|S| = k$ and for all edges $(u,v) \in E$ at most one endpoint is in S.
↳ Either ① $u \notin S$, or ② $v \notin S$, or ③ $u, v \notin S$.



Language $IND = \{\langle(G,k)\rangle \mid G \text{ is a graph with an independent set of size } k\}$

Theorem $3SAT \leq_p IND$.

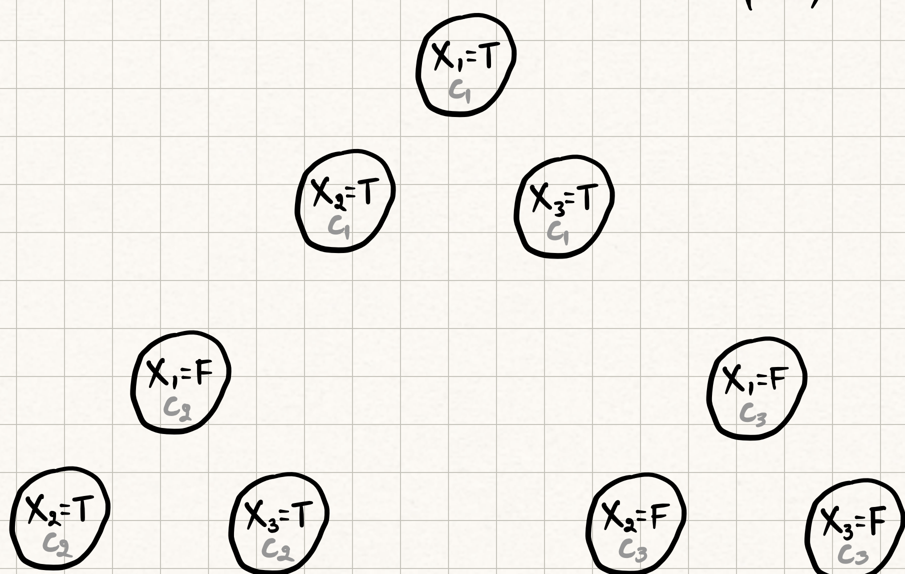
Proof: Intuitively we want to receive a 3SAT valid input ϕ * (it may or may not be satisfiable) and create a graph G such that if there is a k-size independent set we can traslate it to a satisfying assignment for ϕ.

* → is given to you

We need to encode an input of 3SAT to a graph.

example: $\underbrace{(x_1 \vee x_2 \vee x_3)}_{C_1} \wedge \underbrace{(\bar{x}_1 \vee x_2 \vee x_3)}_{C_2} \wedge \underbrace{(\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)}_{C_3}$

(Incorrect) Attempt-1: Introduce a vertex for every literal of every clause where we assign the value that makes it true. For example, for $C_1 = (x_1 \vee x_2 \vee x_3)$

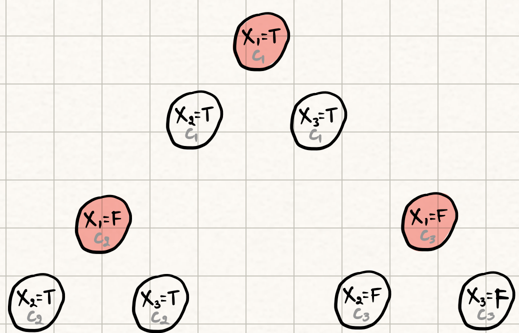


⚠ Every satisfying assignment of ϕ can be translated to an independent set

3SAT ⇒ IND ✓

Not every independent set can be translated to a satisfying assignment

3SAT ⇐ IND ✗



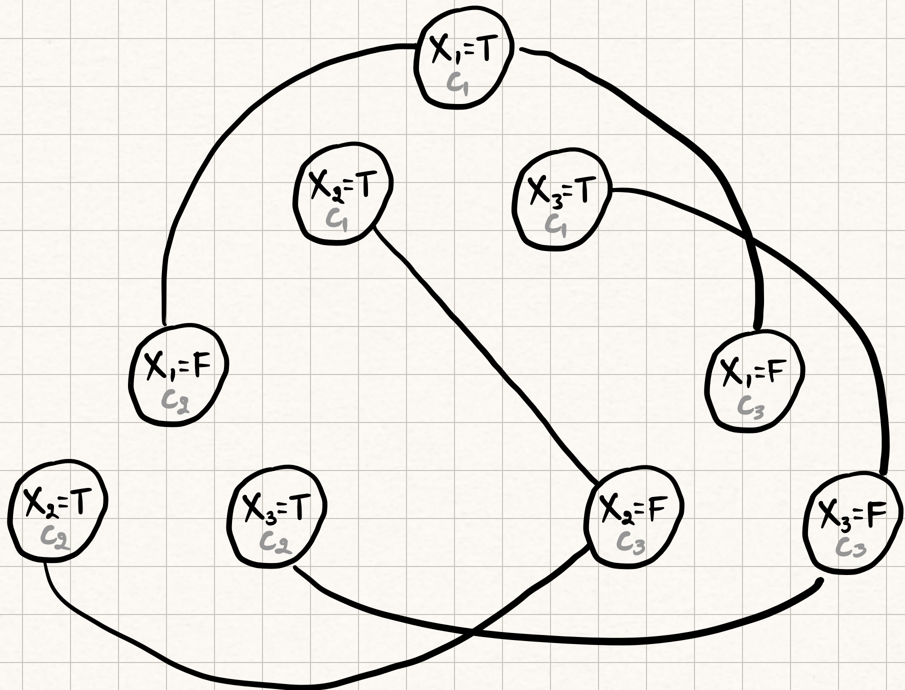
The highlighted set is a independent set of size 3 but cannot give us a satisfying assignment

X_1 cannot be both TRUE and FALSE

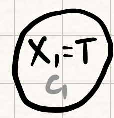
(Incorrect)

Attempt-2: Use the edges of the constructed graph to forbid assignments where the same variable appears as both TRUE and FALSE

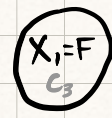
Add an edge if the truth assignments of the same variable are conficting.



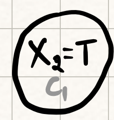
- With the addition of these edges we cannot allow vertices like



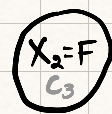
and



to be both in the independent set. Similar argument for



and

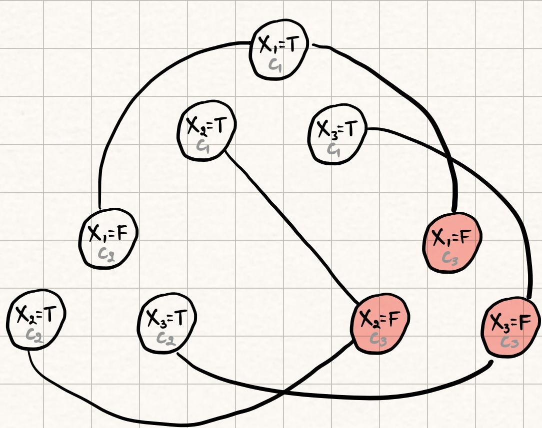


⚠ This improved graph construction is still not what we want. Much like before, every satisfying assignment of ϕ can be translated to an independent set

$3SAT \Rightarrow IND$ ✓

Not every independent set can be translated to a satisfying assignment

$3SAT \Leftarrow IND$ ✗

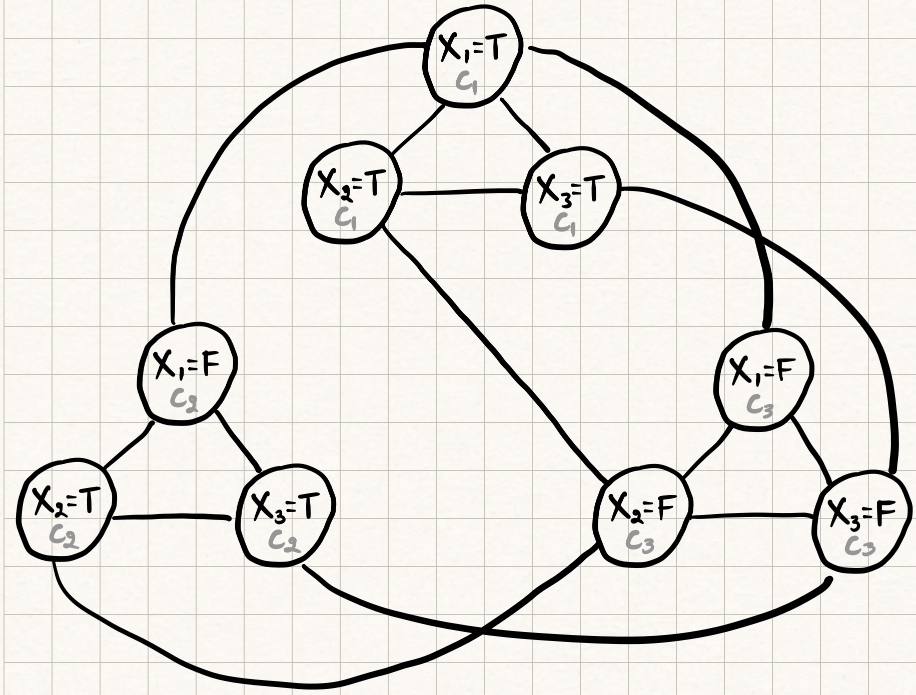


The highlighted set is a independent set of size 3 but cannot give us a satisfying assignment.

The first clause C_1 is not satisfied by this truth assignment.

✱ **Attempt-3**: Add more edges to guarantee that exactly one literal is picked from each clause.

That is, add edges between every per of literals in each clause.



as the number of clauses

If we fix $k=3$, then not only every satisfying assignment gives an independent set
but also
every independent set gives a satisfying assignment.

3SAT $\Rightarrow$ IND ✓
3SAT $\Leftarrow$ IND ✓

A more formal treatment:

The reduction should work for ALL possible inputs

Given any arbitrary 3-SAT input, with n Boolean variables and m clauses, the TM that computes the Karp reduction function outputs a graph $G=(V,E)$.

① Describe the reduction

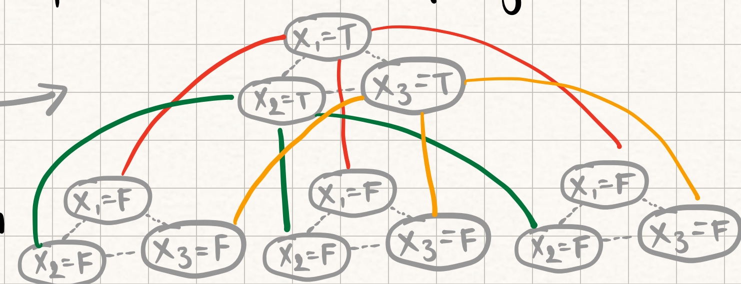
The vertex set of of G is defined as $V=V_1 \cup V_2 \cup \ldots \cup V_m$, where V_j is a group of vertices that has up to three vertices (one vertex per literal) that correspond to literals of the j-th clause. The edge set is defined as $E=E_1 \cup E_2$, where E_1 contains one edge for each pair of vertices that belong to the same clause, and E_2 contains one edge for pair of conflicting vertices (e.g., X_3=T and X_3=F). The Karp reduction assigns $k=m$.

Remember the output of f should be a $\langle (G,k) \rangle$.

② Show that it takes polynomial time

Since we have m clauses, the resulting graph will have at most $3m$ vertices. Since there are m clauses, the edge set E_1 contains at most $3m$ edges while the edge set E_2 contains at most $3\cdot(3m-1)$ edges.

So, overall the Karp reduction creates a graph with $O(m)$ vertices and $O(m)$ edges which takes polynomial time.



③ Show that $\phi \in 3SAT \Rightarrow f(\phi) \in IND$

Given a truth assignment that satisfies ϕ, there must be a literal that is true for each clause. We form the set S from the graph G that was given as an output in the Karp reduction by choosing the vertices as follows:

Iterate through all clauses, for each clause pick a literal that is TRUE (we might have more than one) according to the given truth assignment and choose the corresponding vertex from G as a member of the vertex set S.

- Prove that m is an independent set of size m: S is constructed by choosing exactly one vertex per clause, therefore $|S| = m$. As a next step, we have to show that there are no edges between the vertices in S. Notice that:
  1. Each vertex of S belongs to a different clause-cluster, therefore there are no edges from E_1 between any pair of vertices from S.
  2. The given truth assignment is not conflicting, therefore there are no edges from E_2 between any pair of vertices from S.

Thus, the constructed set S is an independent set of size m.

④ $f(\phi) \in IND \Rightarrow \phi \in 3SAT$

We emphasize here that we are not trying to prove that all independent sets on arbitrary graphs give a truth assignment.

Given an independent set S of size m (from the graph that output by the Karp reduction) we want to find a truth assignment that satisfies ϕ.
Because of the way we constructed G, an independent set of size m must have exactly one vertex from each clause/cluster (due to the E_1 edges).
Additionally, since S is an independent set, there are no E_2 edges that connect vertices in S. Therefore, there are no conflicting truth assignments. Thus, to generate a truth assignment that satisfies ϕ, we pick the literal associated with each node in S and assign either TRUE or FALSE depending on its label in G. ∎