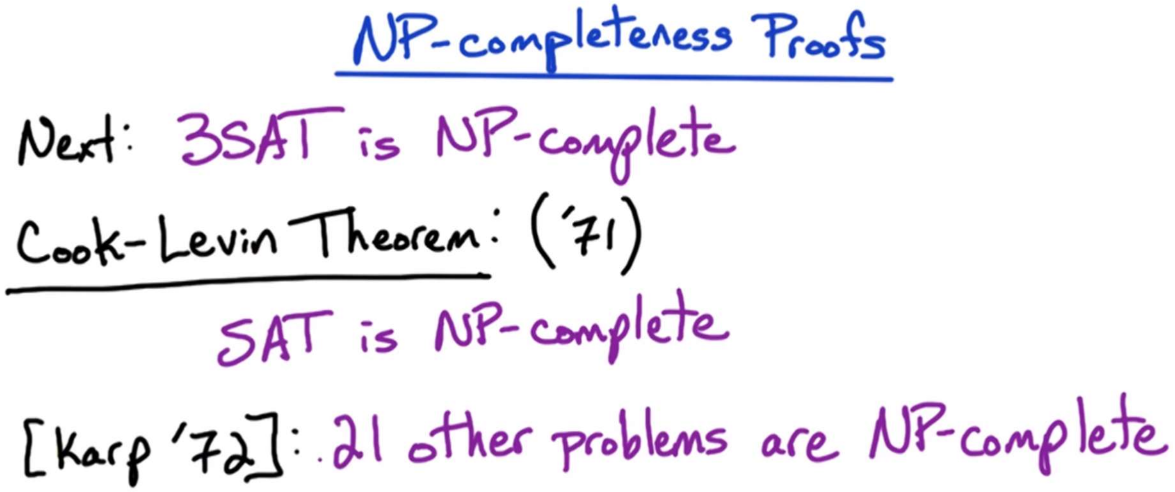
LESSON: NP2: 3SAT

***

(Slide 1) NP-Completeness



NP-completeness Proofs

Next: 3SAT is NP-complete

Cook-Levin Theorem: ('71)

SAT is NP-complete

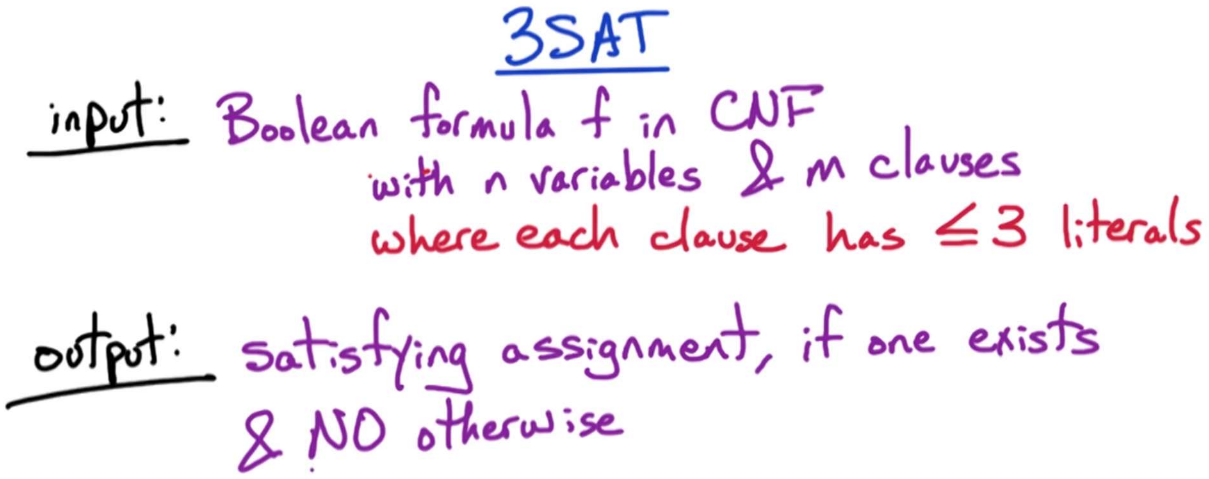
[Karp '72]: 21 other problems are NP-complete

Our plan now is to prove that the 3SAT problem is NP-complete. What we just saw is that if we have a problem such as SAT, which is known to be NP-complete, then this makes our task much easier.

So, we're going to use the fact that SAT is NP-complete. This is known as a Cook-Levin-Theorem. It was proved independently in 1971 by Steven Cook and Leonid Levin. And they proved that SAT is NP-complete. Now, we're going to take this theorem for granted. And later, I'll give you some high-level idea of the proof that the SAT problem is NP-complete. Now, after Steven Cook published his paper, then the importance of NP-completeness was highlighted by a paper by Dick Karp in 1972. He showed 21 other problems which were NP-complete. We're going to look at a few of these 21 in the next few lectures. We'll start with 3SAT, which was one of the 21.

***

(Slide 2) 3SAT



3SAT

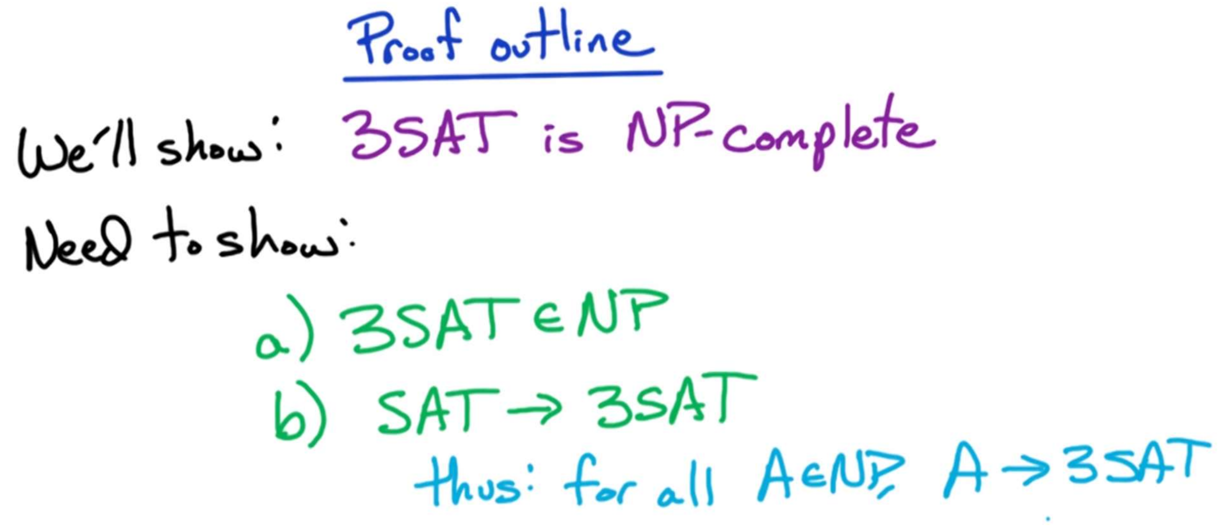
input: Boolean formula f in CNF
with n variables & m clauses
where each clause has ≤ 3 literals

output: satisfying assignment, if one exists
& NO otherwise

Let me give you a quick reminder of the formulation of the 3SAT problem. So the input to the 3SAT problem is a boolean formula f in Conjunctive Normal Form (CNF) and we use n for the number of variables and m for the number of clauses in the formula f. Now the extra restriction in 3SAT as opposed to SAT is that we assume this formula f has the following constraint: Each clause has at most three literals. Now the output for the 3SAT problem is a satisfying assignment if one exists. This is an assignment of True or False to the n variables, so that the formula f evaluates to True. Now if there's no satisfying assignment, we simply output NO.

***

(Slide 3) Proof Outline



Proof outline

We'll show: 3SAT is NP-complete

Need to show:

a) 3SAT ∈ NP

b) SAT → 3SAT

thus: for all A ∈ NP, A → 3SAT

We're going to show now that 3SAT problem is NP complete.

Before we dive into the proof, let's outline what we need to show in order to establish that the 3SAT problem is NP-complete:

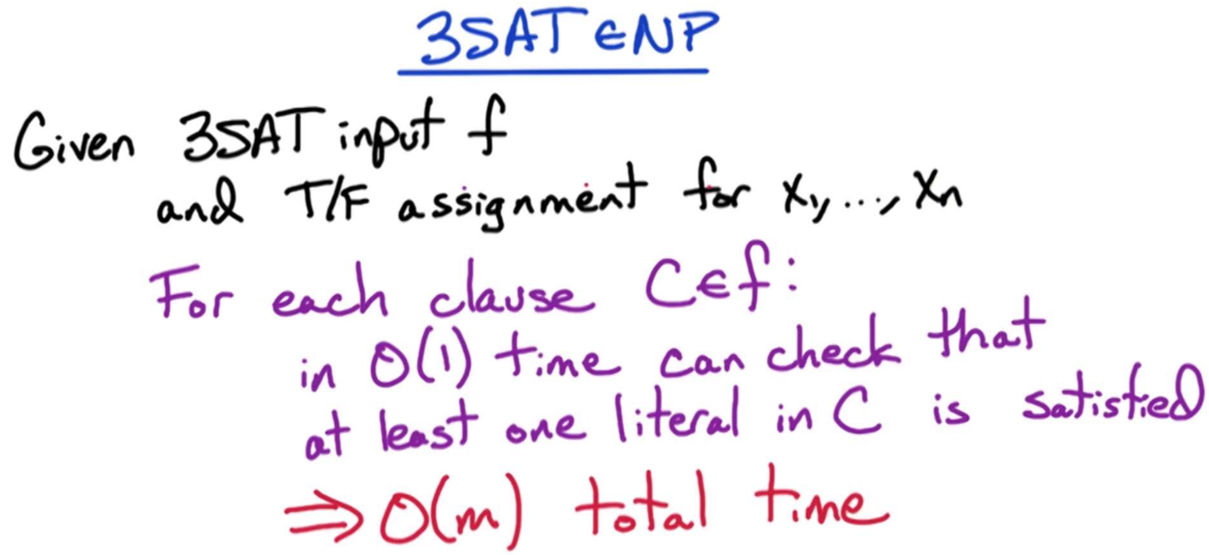
- The first thing we need to show is that the 3SAT problem lies in the class NP. This will be straightforward to prove.
- Now our main task is to take a known NP-complete problem. In this case, all we know is that SAT is NP complete. So we have to just show a reduction from the SAT problem to the 3SAT problem.

Now once we've shown this reduction from SAT to 3SAT, what does that establish? That establishes that for every problem in NP, we have a reduction from this problem A to SAT. And then we have this reduction from SAT to 3SAT, therefore, we have a reduction from A to 3SAT. The implication of this is that if we have a polynomial time algorithm for 3SAT, then we have a polynomial time algorithm for every problem in NP, because we can reduce every problem in NP to 3SAT.

So let's start with the easy task. Let's prove that 3SAT is in the class NP.

***

(Slide 4) 3SAT in NP



3SAT $\in$ NP

Given 3SAT input f
and T/F assignment for $x_1, \dots, x_n$

For each clause $C \in f$:
in $O(1)$ time can check that
at least one literal in C is satisfied
$\Rightarrow O(m)$ total time

Now, to prove that 3SAT is in the class NP, we have to show that we can verify solutions efficiently.

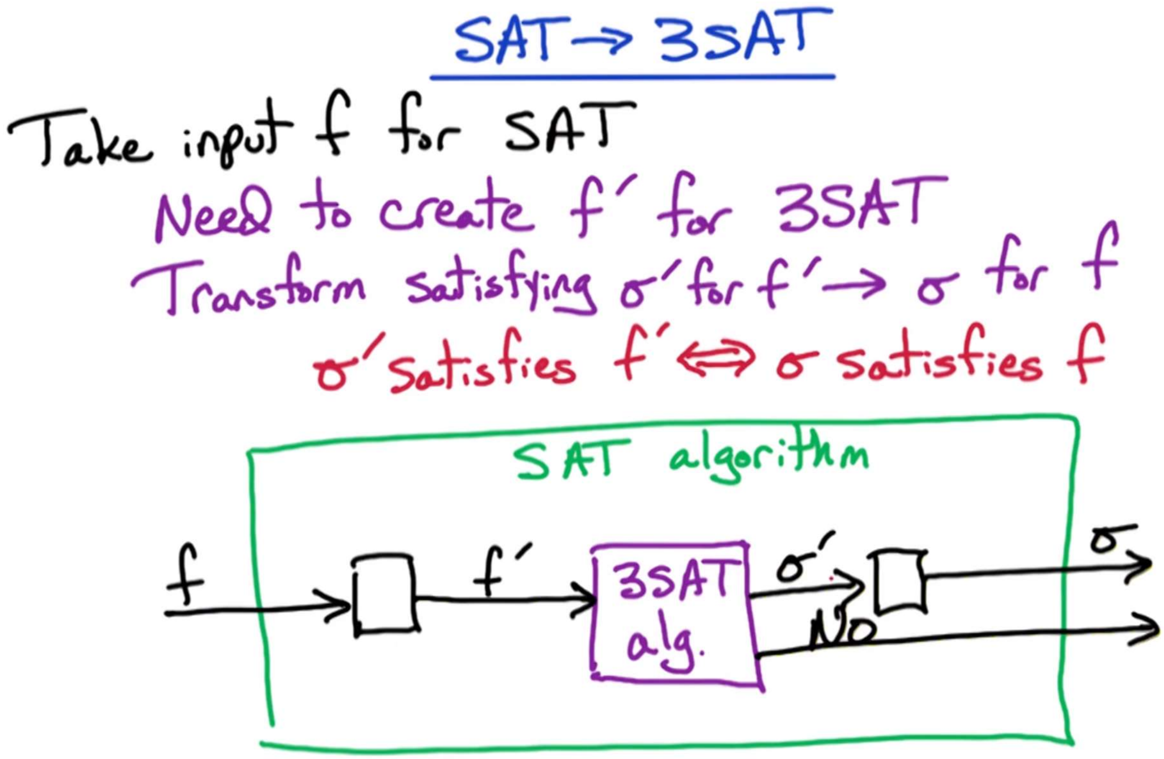
So, let's take a particular input for the 3SAT problem. So f is our input for the 3SAT problem. And let's take a proposed solution. So, this is a True/False assignment for the n variables. Now, we need to check that this assignment is a satisfying assignment for this formula. How are we going to verify that this assignment is a satisfying assignment? Well, we're going to go through the clauses.

So let's take each clause. For a particular clause C, it will take us O(1) time to check that at least one of the literals in C is satisfied. Why is it O(1) time? Because each clause has at most three literals.

Now, if every clause C is satisfied, then the formula f is satisfied. It takes O(1) time per clause. There's m clauses, so it takes O(m) total time to verify that this assignment satisfies the formula. So this proves that the 3SAT problem is in the class NP.

***

(Slide 5) SAT 3SAT



SAT → 3SAT

Take input f for SAT

Need to create f′ for 3SAT

Transform satisfying σ′ for f′ → σ for f

σ′ satisfies f′ ⟺ σ satisfies f

Now, let's look at the task of reducing SAT to 3SAT.

Let's outline first what we need to prove. Now, we're assuming we have an algorithm a polynomial time algorithm for the 3SAT problem, and we're going to use this as a blackbox. And we're going to construct an algorithm for the SAT problem, using this 3SAT algorithm as a subroutine. So, what do we need to do?

- We need to take an input for the SAT problem. So we have an f, which is an input formula for the SAT problem. Then we have to transform this input f into an input for the 3SAT problem. We'll use f’ (f prime) to denote the input to 3SAT problem.

  Now this is a bit tricky to do. Why? Because f might have some big clauses. It might have some clauses which contain maybe n literals. But our input for 3SAT has to have clauses of size at most three. So somehow we have to transform these big clauses, into a series of small clauses. And we need to do it in such a way that if we have a satisfying assignment Σ’ (Sigma prime), which is a satisfying assignment for f’, our 3SAT formula, then we can transform this satisfying assignment for the 3SAT input into a satisfying assignment sigma for the original SAT input.

Moreover, we want that if our 3SAT formula, f′, has no satisfying assignments (So, there's no Sigma prime – So, the algorithm is going to output No) , we want that our original SAT formula f also has no satisfying assignment. So you want that f′ has no satisfying assignment if and only if f has no satisfying assignment. And that way we can simply output NO, in both cases.

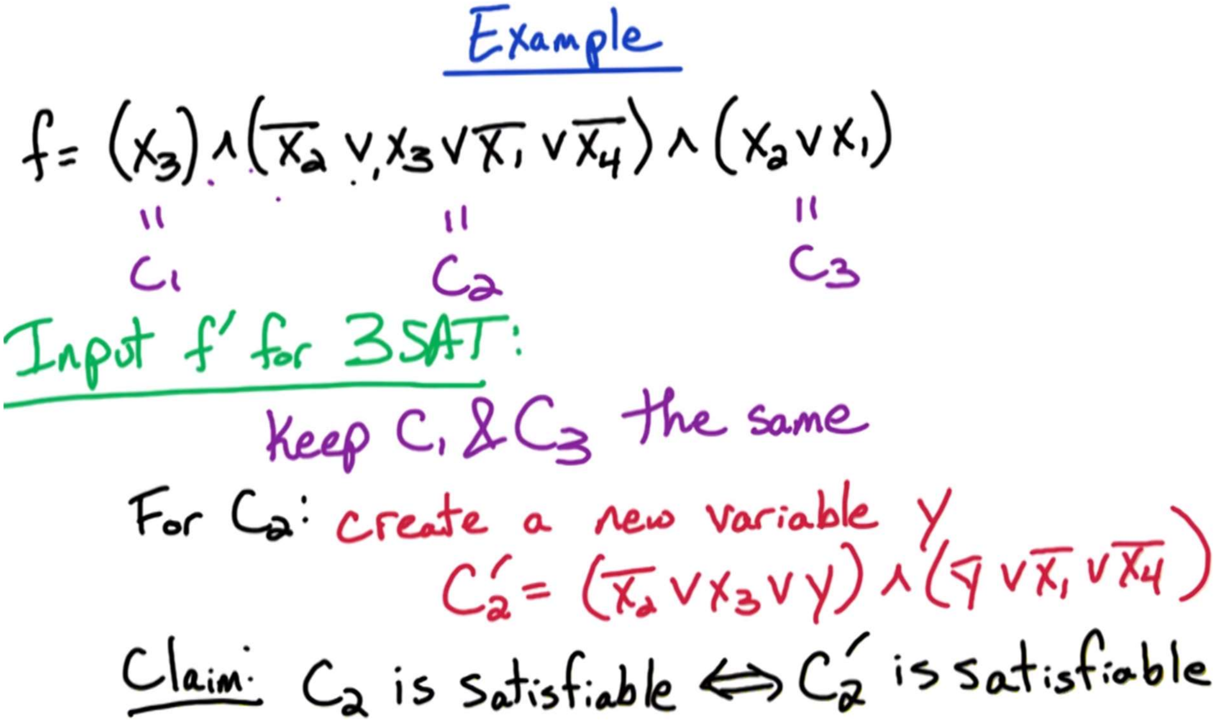
So once again what do we need to do? We need to take an input f for the SAT problem, and we need to create an input for the 3SAT problem, and then given a satisfying assignment Σ′ for the 3SAT input, we need to transform it and make a satisfying assignment, for the SAT formula. And we need that Σ′ satisfies f′. So this is a satisfying assignment for f′ (the 3SAT input) if and only if this transformed output Σ is a satisfying assignment for our original SAT formula.

So we have a satisfying assignment for the 3SAT input if and only if we have a satisfying assignment for the SAT formula. Why do we need this equivalence? Because we need that no instances for the 3SAT formula correspond to no instances for the SAT formula. So if we get a NO output, we can output a NO for the SAT input.

Now this is the main test, so let's dive into it. We want to take an input f for the SAT problem, and transform it into a valid input for the 3SAT problem.

***

(Slide 6) Example



Example

$$f = \underset{C_1}{\underset{\shortparallel}{(x_3)}} \wedge \underset{C_2}{\underset{\shortparallel}{(\overline{x_2} \vee x_3 \vee \overline{x_1} \vee \overline{x_4})}} \wedge \underset{C_3}{\underset{\shortparallel}{(x_2 \vee x_1)}}$$

Input f′ for 3SAT:

Keep C_1 & C_3 the same

For C_2: create a new variable y

$$C_2' = (\overline{x_2} \vee x_3 \vee y) \wedge (\overline{y} \vee \overline{x_1} \vee \overline{x_4})$$

Claim: C_2 is satisfiable $\Leftrightarrow$ C_2' is satisfiable

Let's try to get some idea for the reduction.

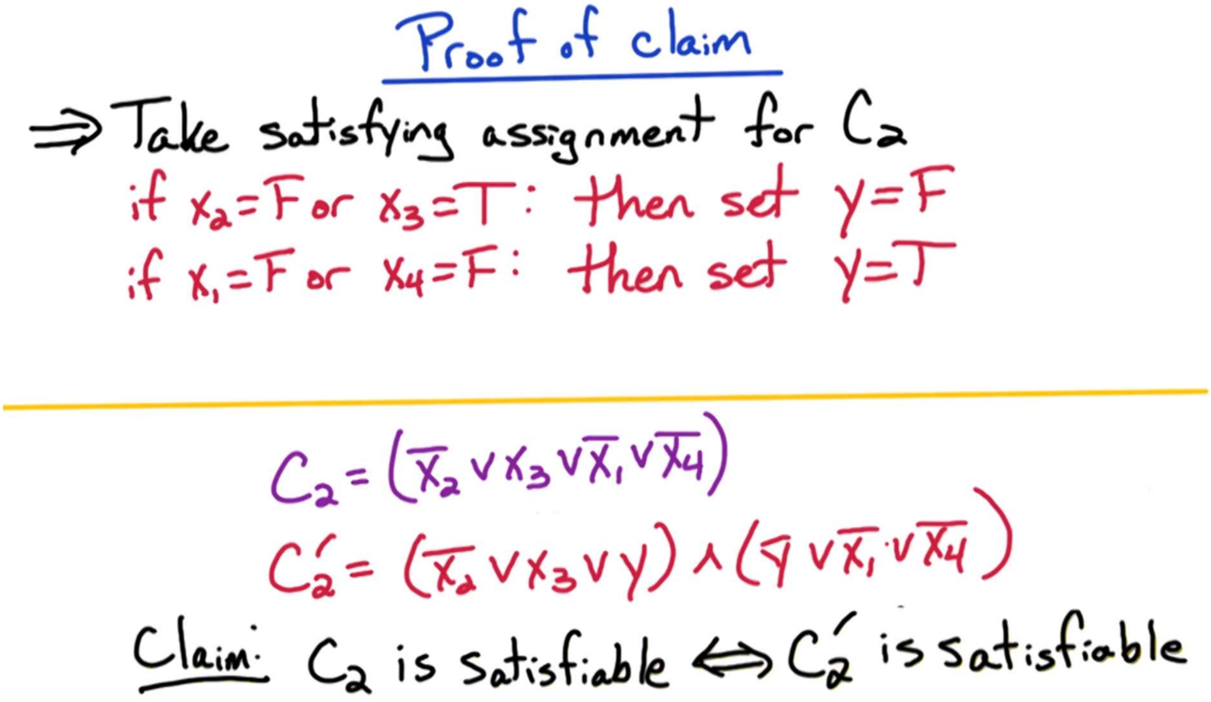
So let's take a sample input for the SAT problem. So here's a formula with four variables and three clauses. Let's label the clauses as c1, c2, and c3. Now, let's define our input for the 3SAT problem, let's denote this as f′, where clause 1 and clause 3. we can keep the same. Those are valid inputs. Those are valid clauses for a 3SAT.

Now, the challenge is what do we do with the second clause. This is a clause of size four, but valid inputs with 3SAT have size at most three. So, what are we going to do for this second clause? Well, we're going to create a new variable y and we're going to replace the second clause by the following pair of clauses: $(\overline{x2} \vee x3 \vee y) \wedge (\overline{y} \vee \overline{x1} \vee \overline{x4})$. So it's $(\overline{x2} \vee x3 \vee y)$; these are the first two literals in c2. And the second clause is $(\bar{y} \vee \overline{x1} \vee \overline{x4})$; these are the last two literals in the clause c2.

Now, the key claim is that this original clause c2 of size four is satisfiable if and only if this new pair of clauses is satisfiable. So c2 is satisfiable if and only if c2′ is satisfiable. Therefore, replacing c2 by this pair of clauses makes an equivalent formula. And. in this new formula, all the clauses have size 3.

***

(Slide 7) Claim: Forward



Proof of claim

⇒ Take satisfying assignment for C_2

if $x_2=F$ or $x_3=T$: then set $y=F$

if $x_1=F$ or $x_4=F$: then set $y=T$

$C_2=(\overline{x}_2 \vee x_3 \vee \overline{x}_1 \vee \overline{x}_4)$

$C_2'=(\overline{x}_2 \vee x_3 \vee y) \wedge (\overline{y} \vee \overline{x}_1 \vee \overline{x}_4)$

Claim: C_2 is satisfiable ⇔ C_2' is satisfiable

Now, let's go ahead and prove this claim, so that we have some idea how to generalize this construction.

Here's our original clause of size 4: c2 = ($\overline{x2}$ ∨ x3 ∨ $\overline{x1}$ ∨ $\overline{x4}$) and here's our new pair of clauses each of size at most three: c2′ = ($\overline{x2}$ ∨ x3 ∨ y) ∧ ($\bar{y}$ ∨ $\overline{x1}$ ∨ $\overline{x4}$) . And we want to prove that this original clause c2 is satisfiable, if and only if this new pair of clauses c2′ is satisfiable.

Let's do the forward direction first. Let's take a satisfying assignment for c2. This is an assignment for x2, x3, x1, and x4, which satisfies this clause and we want to show that there is a satisfying assignment for c2′. Now in order for this assignment to satisfy c2, one of the four cases must hold: Either:

1. x_2 equals False
2. x_3 equals True
3. x_1 equals False
4. x_4 equals False.

Maybe some combination of those hol; but, at least one of those has to hold. We'll break up these four cases into the following pair of cases:

- We'll consider whether x2 equals False or x3 equals True. These are the pair of literals which appear in the first clause of c2′

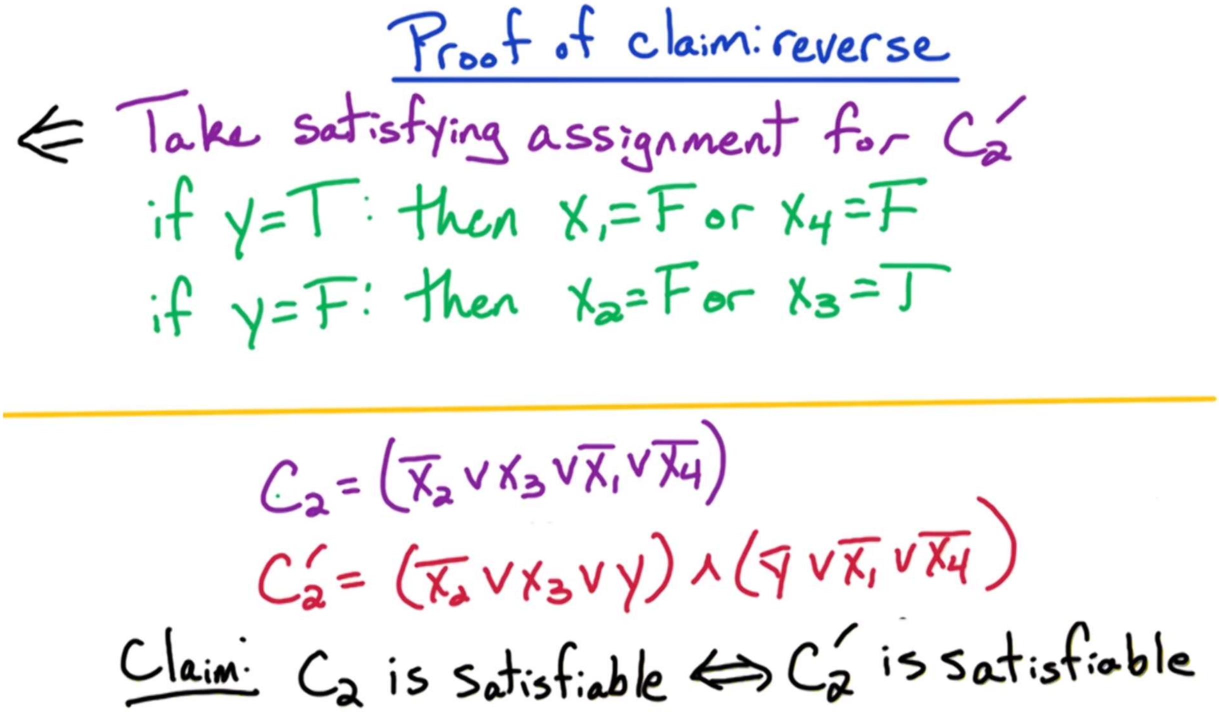
- and the other case is if x1 equals False or x4 equals False. This is the pair of literals which appear in the second clause of c2'.

Now if x2 = False or x3 = True, then we're going to set y = False. Notice, in this case, we have x2 = False or x3 = True. Therefore, the first clause is satisfied and we're setting y = False. So this satisfies the second clause. c2' is satisfied in this case.

Similarly in the second case, if x1 = False or x4 = False, then we set y = True. In this case, since y is true, this satisfies the first clause. And since x1 is False or x4 is False, this satisfies the second clause. And given a satisfying assignment for c2, we've constructed a satisfying assignment for c2'.

***

(Slide 8) Claim: Reverse



Proof of claim: reverse

$\Leftarrow$ Take satisfying assignment for C_2'

if $y=T$: then $x_1=F$ or $x_4=F$

if $y=F$: then $x_2=F$ or $x_3=T$

$C_2 = (\overline{x_2} \vee x_3 \vee \overline{x_1} \vee \overline{x_4})$

$C_2' = (\overline{x_2} \vee x_3 \vee y) \wedge (\overline{y} \vee \overline{x_1} \vee \overline{x_4})$

Claim: C_2 is satisfiable $\Leftrightarrow$ C_2' is satisfiable

Now let's look at the reverse implication. In this case we're going to take a satisfying assignment for c2′ and we're going to construct a satisfying assignment for c2. What we're going to show is that if we just ignore y, whatever the assignment was for x1, x2, x3, and x4 which satisfied c2′, that's going to satisfy c2.

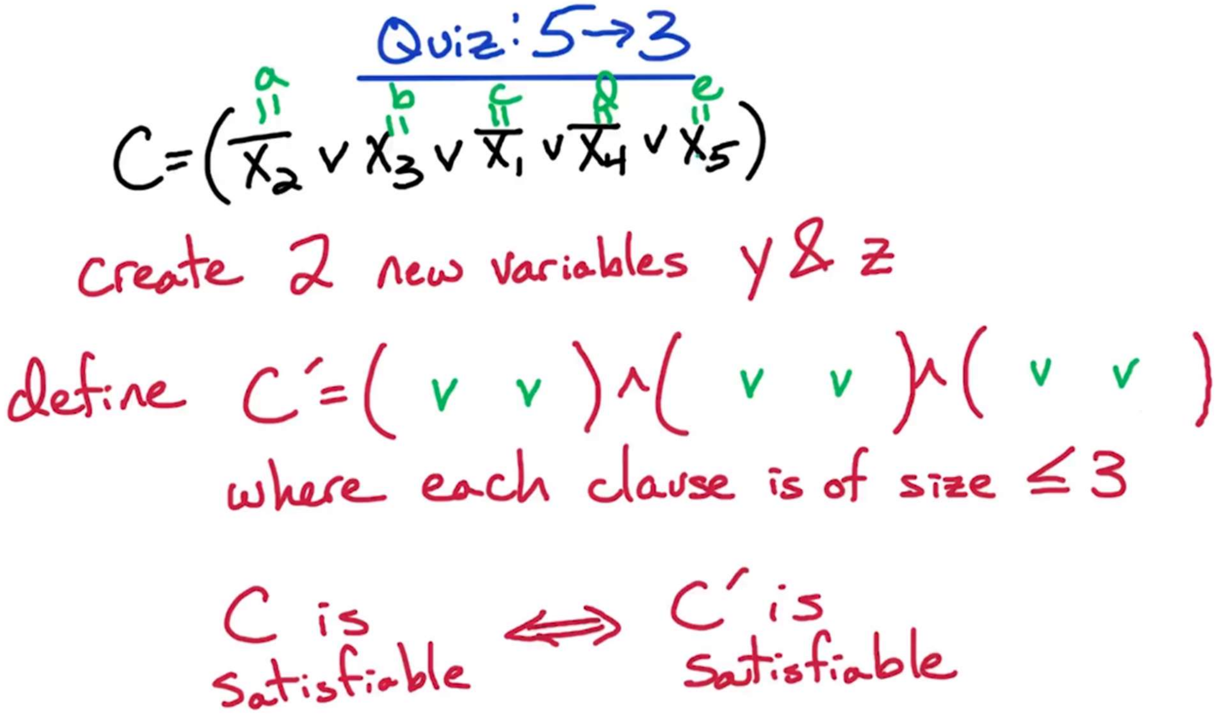
Now there will be two cases: either y = True or y = False:

- Suppose y = True. We know that c2 is satisfied, so each of these two clauses in c2′ is satisfied. Y is True so that satisfies the first clause. How about the second clause? Well, either x1 must be False or x4 must be False. Well, if x1 is False or x4 is False, both of these satisfy c2. So, in either of these scenarios, we have an assignment which satisfies c2.
- Now suppose y = False. So this assignment for y satisfies this second clause of c2′. How do we satisfy this first clause of c2′? What we're assuming that this is a satisfying assignment for c2′ so it must satisfy this first clause. y = False, so that's not helping. So it must be the case that either x2 = False or x3 = True.. Now if x2 = False, that means that this is an assignment which satisfies c2 and if x3 = True, then also this is an assignment which satisfies c2.

So in any of these four cases (where y = T, and x1 = F or x4 = F; or y = F, and x2 = F or x3 = T), we have an assignment which satisfies c2. So every satisfying assignment for c2′ if we ignore y, it's also a satisfying assignment for c2. And this establishes the reverse implication. We've done both directions, so we've shown the if and only if.

***

(1st Slide 9) Quiz: 5-SAT 3-SAT



Quiz: 5→3

$C=(\overset{a}{\overline{x_2}} \vee \overset{b}{x_3} \vee \overset{c}{\overline{x_1}} \vee \overset{d}{\overline{x_4}} \vee \overset{e}{x_5})$

create 2 new variables y & z

define $C' = (\quad \vee \quad \vee \quad) \wedge (\quad \vee \quad \vee \quad) \wedge (\quad \vee \quad \vee \quad)$

where each clause is of size ≤ 3

C is satisfiable ⟺ C′ is satisfiable

Now, we just saw how to transform a clause of size 4 into a pair of clauses of size 3. Now, let's try to generalize this idea. Let's do it one step at a time. So let's take a clause of size 5 now: C = ($\overline{x2}$ ∨ x3 ∨ $\overline{x1}$ ∨ $\overline{x4}$ ∨ x5), and let's try to transform this into a triple of clauses of size 3

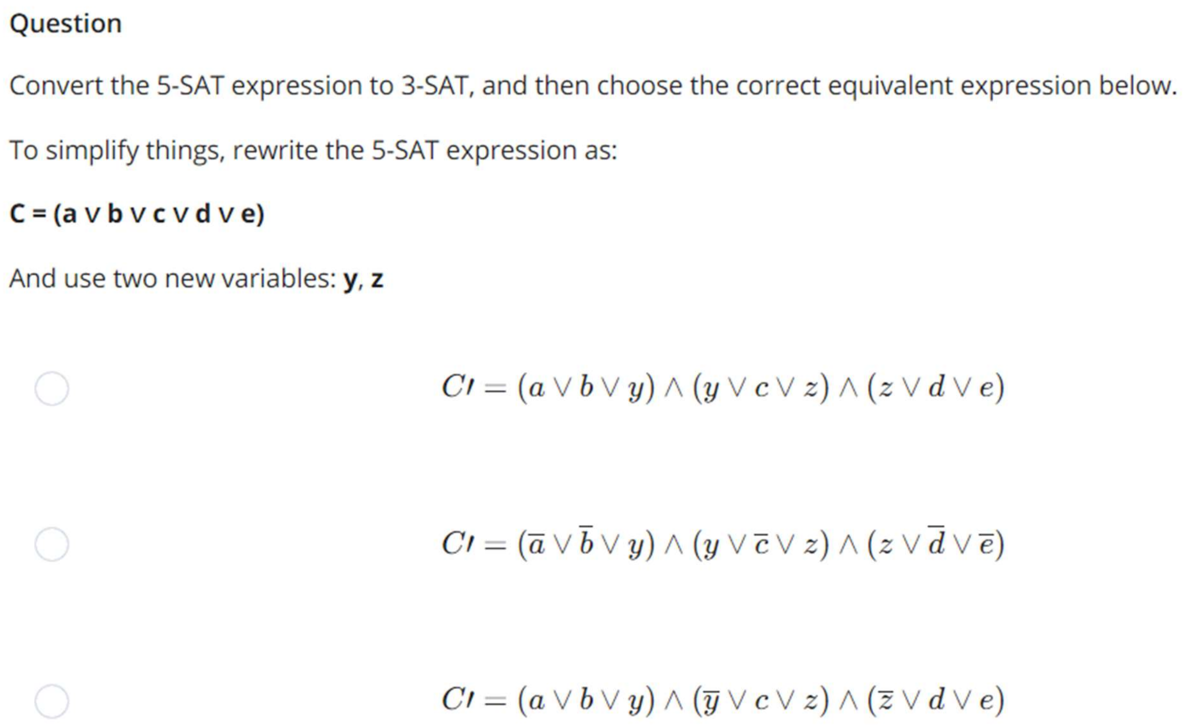
Now, previously, to transform the clauses size 4 into a pair of clauses of size 3, we added one new variable y. Now, to transform a clause of size 5 into triple clauses of size 3, we're going to create two new variables, y and z. Now, we're going to make a formula, C′, which is three clauses, and each of these clauses is going to be of size at most 3. Actually, they're going to be of size equal to 3. And we want to do this in such a way that C is satisfiable. The original clause C is satisfiable if and only if this triple of clauses C′ is satisfiable. And, therefore, in our original input F, if we have a clause such as this, we can replace it by this triple of clauses, and we'll get an equivalent formula.

Now, the quiz is to define C′. Define this triple of clauses. Each is of size exactly three literals, and use this pair of new variables, y and z. For simplicity, for entering your solution, let me enter the OR symbols for you. And also to simplify your input, let's relabel these literals. So let's relabel these five literals as a, b, c, d, e. So a is equal to $\overline{x2}$, b is equal to x3, c is equal to $\overline{x1}$ bar, d is equal to $\overline{x4}$, e is equal to x5. So, use a, b, c, d, e, and y and z, and create a formula, C′ prime such that C′ is satisfiable if and only if C is satisfiable.

***

NP2: Quiz: 5-SAT 3-SAT

See Slide 9 above.



Question

Convert the 5-SAT expression to 3-SAT, and then choose the correct equivalent expression below.

To simplify things, rewrite the 5-SAT expression as:

C = (a ∨ b ∨ c ∨ d ∨ e)

And use two new variables: **y**, **z**

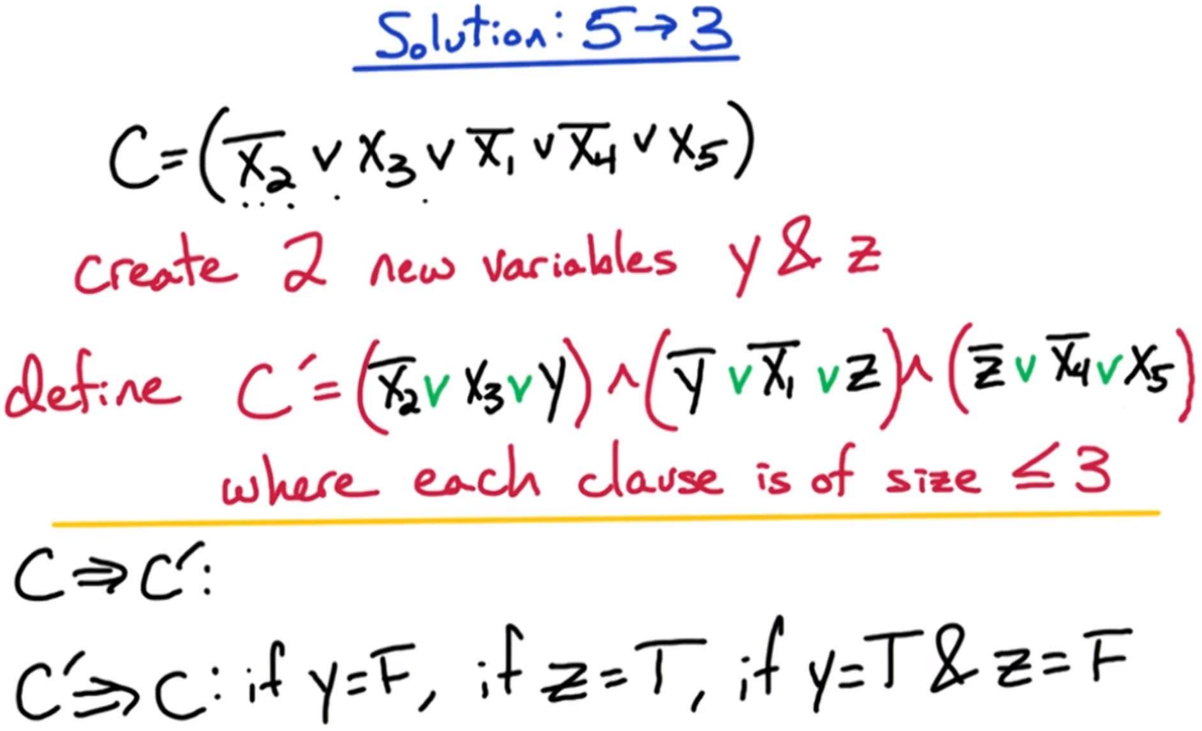
○ $C\prime = (a \vee b \vee y) \wedge (y \vee c \vee z) \wedge (z \vee d \vee e)$

○ $C\prime = (\overline{a} \vee \overline{b} \vee y) \wedge (y \vee \overline{c} \vee z) \wedge (z \vee \overline{d} \vee \overline{e})$

○ $C\prime = (a \vee b \vee y) \wedge (\overline{y} \vee c \vee z) \wedge (\overline{z} \vee d \vee e)$

***

(2nd Slide 9) Quiz: 5-SAT 3-SAT (Answer)



Solution: 5→3

$C=(\overline{x_2} \vee x_3 \vee \overline{x_1} \vee \overline{x_4} \vee x_5)$

create 2 new variables y & z

define $C'=(\overline{x_2} \vee x_3 \vee y) \wedge (\overline{y} \vee \overline{x_1} \vee z) \wedge (\overline{z} \vee \overline{x_4} \vee x_5)$

where each clause is of size ≤ 3

$C \Rightarrow C'$:

$C' \Rightarrow C$: if $y=F$, if $z=T$, if $y=T$ & $z=F$

The solution is the following: $C' = (\overline{x2} \vee x3 \vee y) \wedge (\bar{y} \vee \overline{x1} \vee z) \wedge (\bar{z} \vee \overline{x4} \vee x5)$

- I take the first two literals of C, and y is a new variable. $(\overline{x2} \vee x3 \vee y)$ is the first clause.
- The second clause is the following, I take $(\bar{y} \vee \overline{x1} \vee z)$.
- The last clause is $(\bar{z} \vee \overline{x4} \vee x5)$; x4 is the fourth literal, and x5 is the fifth literal.

Instead of formally proving this now, let me give you a quick idea of the proof and then we'll do the general construction and then we'll prove the general construction is correct.

C => C′:

Let's suppose that we have a satisfying assignment for $C = (\overline{x2} \vee x3 \vee \overline{x1} \vee \overline{x4} \vee x5)$ and let's see how we construct a satisfying assignment for $C' = (\overline{x2} \vee x3 \vee y) \wedge (\bar{y} \vee \overline{x1} \vee z) \wedge (\bar{z} \vee \overline{x4} \vee x5)$.

So, one of these five literals (in C) must be satisfied. Let's suppose the middle one, $\overline{x1}$ is satisfied, so x1 is set to false – so, that's going to satisfy the second clause. How do we satisfy the first and third clause? Well, here we use these auxiliary variables, setting y = True satisfies the first clause and setting Z = False satisfies the last clause. And, in general, one of these five is going to be satisfied, that's going to satisfy one of these three clauses. And then we can use these other two auxiliary variables to satisfy the other two clauses.

(C′ => C):

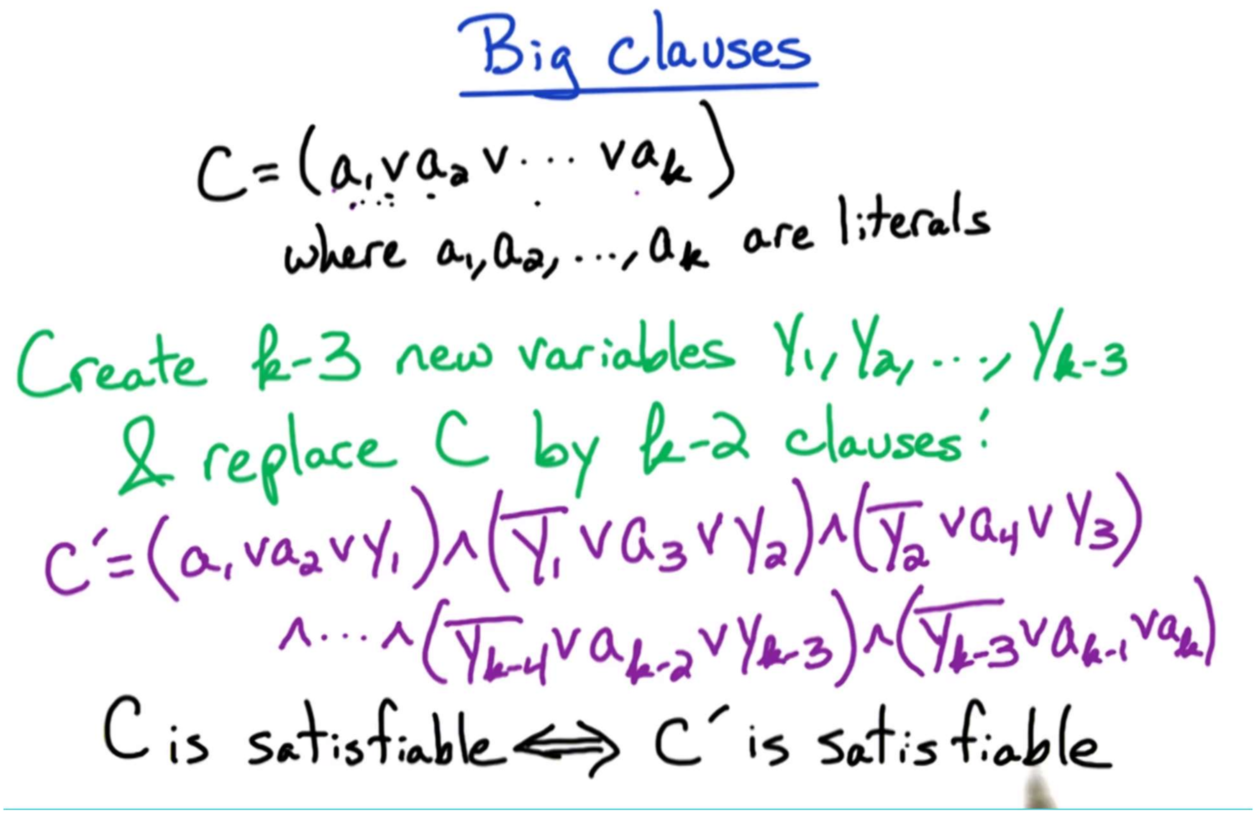
How about the reverse implication? Suppose we have a satisfying assignment for C′. How do we get a satisfying assignment for C?

- Well, take the case where y = False. So the literal y in the first clause of C′ is not satisfied so, either $\overline{x2}$ or x3 must be satisfied. If one of these two literals is satisfied, they're satisfied here (in C) as well, so C is satisfied.
- Now, suppose that z is set to true, then this literal $\bar{z}$ in the third clause of C′ is not satisfied. So, one of these two literals in the third clause (either $\overline{x4}$ or x5) must be satisfied. Either x4 is set to False or x5 is set to True. In which case, one of these last two literals of C is satisfied.
- The last case is a complement of these two. y is set to True and Z is set to False. Well, since we're taking a satisfying assignment for C′, look at how this second clause of C′ can be satisfied. Y is set to True and Z is set to false so these two literals ($\bar{y}$, z) are not satisfied. So the only way to satisfy this clause is to set x1 = False. So if x1 is set to False, then we satisfy this third literal in C.

Now, let's do the general construction. Let's say we have a clause of size k. Now we're going to create a series of clauses which is equivalent to this original clause. Now, when we had a clause of size 4, we added one new variable and we had a pair of clauses. When we had a clause of size 5, we created two new variables and then we have a triple of clauses. Now, if we have a clause of size k, we're going to create a k-3 new variables and we're going to create k-2 clauses. So, let's go ahead and do the general construction.

***

(Slide 10) Big Clauses



Big clauses

$C = (a_1 \vee a_2 \vee \cdots \vee a_k)$

where $a_1, a_2, \ldots, a_k$ are literals

Create k-3 new variables $y_1, y_2, \ldots, y_{k-3}$

& replace C by k-2 clauses:

$$C' = (a_1 \vee a_2 \vee y_1) \wedge (\overline{y}_1 \vee a_3 \vee y_2) \wedge (\overline{y}_2 \vee a_4 \vee y_3)$$
$$\wedge \cdots \wedge (\overline{y}_{k-4} \vee a_{k-2} \vee y_{k-3}) \wedge (\overline{y}_{k-3} \vee a_{k-1} \vee a_k)$$

C is satisfiable $\Longleftrightarrow$ C′ is satisfiable

Let's consider now a general clause C of size k, and let's label the literals in this clause by a1, a2 up to ak: C = a1 ∨ a2 ∨ ··· ∨ ak. Now, for this clause, we're going to create k-3 new variables.

Recall that when k was 4, we created one new variable. When k was 5, we created two new variables. In general, we're going to create k-3 new variables. Let's label these new variables as y1 through y_{k-3}.

Now it's important to note that every clause of size greater than 3 creates new variables, and these new variables are distinct for each clause. So, for each clause, we might create O(n) new variables. There's m clauses so we might have O(nm) new variables in total.

Now we're going to replace this original clause C, by the following k-2 clauses.

First clause: (a1 ∨ a2 ∨ y1): So we take the first two literals, a1, a2, and the first new variable y1, and the first clause is (a1 ∨ a2 ∨ y1)

Second clause: ($\overline{y1}$ ∨ a3 ∨ y2): and we use the negative of the first new variable, the third literal a3, and then the positive of the next new variable.

And this gives us our pattern.

Third clause: $(\overline{y2} \vee a4 \vee y3)$: So, we then use $\overline{y2}$ - the negative of the second variable, and then we use the next literal, a4, and then we use the next new variable, positive form, y3.

Now we continue this pattern, and then the last two clauses look as follows:

Next to last clause: $(\bar{y}_{k-4} \vee a_{k-2} \vee y_{k-3})$: So the penultimate clause looks as follows. It's going to have $\bar{y}_{k-4} \vee a_{k-2}$ - this is the third to last literal - $\vee\ y_{k-3}$ is the last new variable in the positive form. This penultimate clause follows the same pattern.
Last clause: $(\bar{y}_{k-3} \vee a_{k-1} \vee ak)$: The last clause is going to be slightly different. It has y_{k-3} same pattern as before and then we use the last two literals of C. So we have $\bar{y}_{k-3} \vee a_{k-1} \vee ak$.

This defines the formula C′:
$C' = (a1 \vee a2 \vee y1) \wedge (\overline{y1} \vee a3 \vee y2) \wedge (\overline{y2} \vee a4 \vee y3) \wedge \cdots \wedge (\bar{y}_{k-4} \vee a_{k-2} \vee y_{k-3}) \wedge (\bar{y}_{k-3} \vee a_{k-1} \vee ak)$

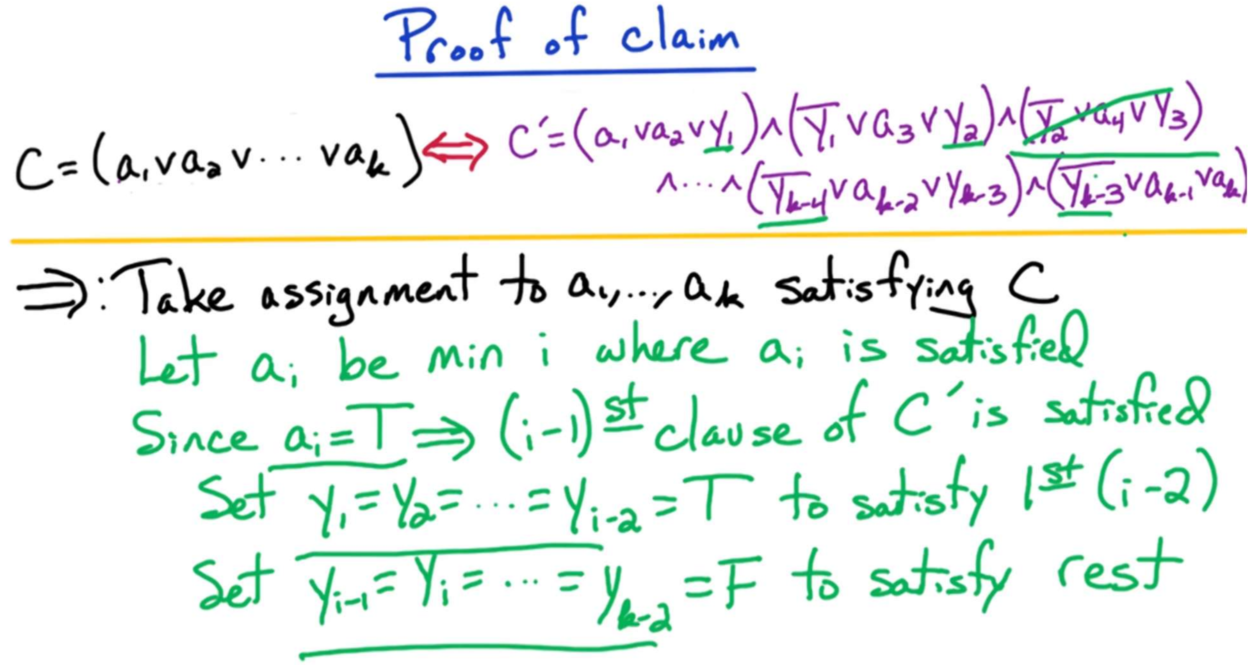
Now our claim is that the original clause C is satisfiable if and only if this new sequence of clauses, C′, is satisfiable.

Now, for our original input to the SAT problem, for every clause which is size bigger than size 3, we can replace it under this following construction: We take this clause, which is size bigger than three; we replace it by this sequence of clauses which are all of size exactly equal to three; and, then we get a valid input to the 3SAT formula. And, the key is that this new formula is satisfiable if and only if the original formula is satisfiable, within the equivalent formula.

So let's go ahead and prove this claim, that C is satisfiable if and only if C′ is satisfiable. And, then we'll be pretty much done with the reduction.

***

(Slide 11) General Claim: Forward



Proof of claim

$C=(a_1 \vee a_2 \vee \cdots \vee a_k) \Leftrightarrow C'=(a_1 \vee a_2 \vee \underline{y_1}) \wedge (\bar{y}_1 \vee a_3 \vee \underline{y_2}) \wedge (\cancel{\bar{y}_2} \vee a_4 \vee y_3)$
$\wedge \cdots \wedge (\underline{\bar{y}_{k-4}} \vee a_{k-2} \vee y_{k-3}) \wedge (\underline{\bar{y}_{k-3}} \vee a_{k-1} \vee a_k)$

$\Rightarrow$: Take assignment to $a_1, \ldots, a_k$ satisfying C

Let a_i be min i where a_i is satisfied

Since $\underline{a_i=T} \Rightarrow (i-1)^{st}$ clause of C' is satisfied

Set $y_1=y_2=\cdots=y_{i-2}=T$ to satisfy 1^{st} $(i-2)$

Set $\underline{\overline{y_{i-1}=y_i=\cdots=}\,y_{k-2}}=F$ to satisfy rest

This was our construction:

C = : C = a1 ∨ a2 ∨ ··· ∨ ak

<=>

C′ = (a1 ∨ a2 ∨ y1) ∧ ($\overline{y1}$ ∨ a3 ∨ y2) ∧ ($\overline{y2}$ ∨ a4 ∨ y3) ∧ ··· ∧ ($\bar{y}_{k-4}$ ∨ a_{k-2} ∨ y_{k-3}) ∧ ($\bar{y}_{k-3}$∨ a_{k-1} ∨ ak)

We took this clause of size k. So we took C which was a_1 ∨ a_2 ∨…up to a_k and we defined this formula C′, which consisted of k-2 clauses. And our claim is that C is satisfiable if and only if C′ is satisfiable. Now, let's prove this claim.

C => C′:

Let's start with the forward implication. So, let's take an assignment to these literals which satisfies this clause C and let's prove that there is a satisfying assignment to C′.

Now, in order for this assignment to satisfy C, one of these literals (a1,a2,…,ak) must be satisfied. Let's let ai be the first satisfied literal. So, let ai be the first literal satisfied. Now, if ai is satisfied that's going to satisfy one of these clauses. If i = 1, a1, that's in the first clause. And, if i is at least two, then ai appears in the i-1st clause. So, this literal ai being set to true satisfies the i-1st clause of C′.

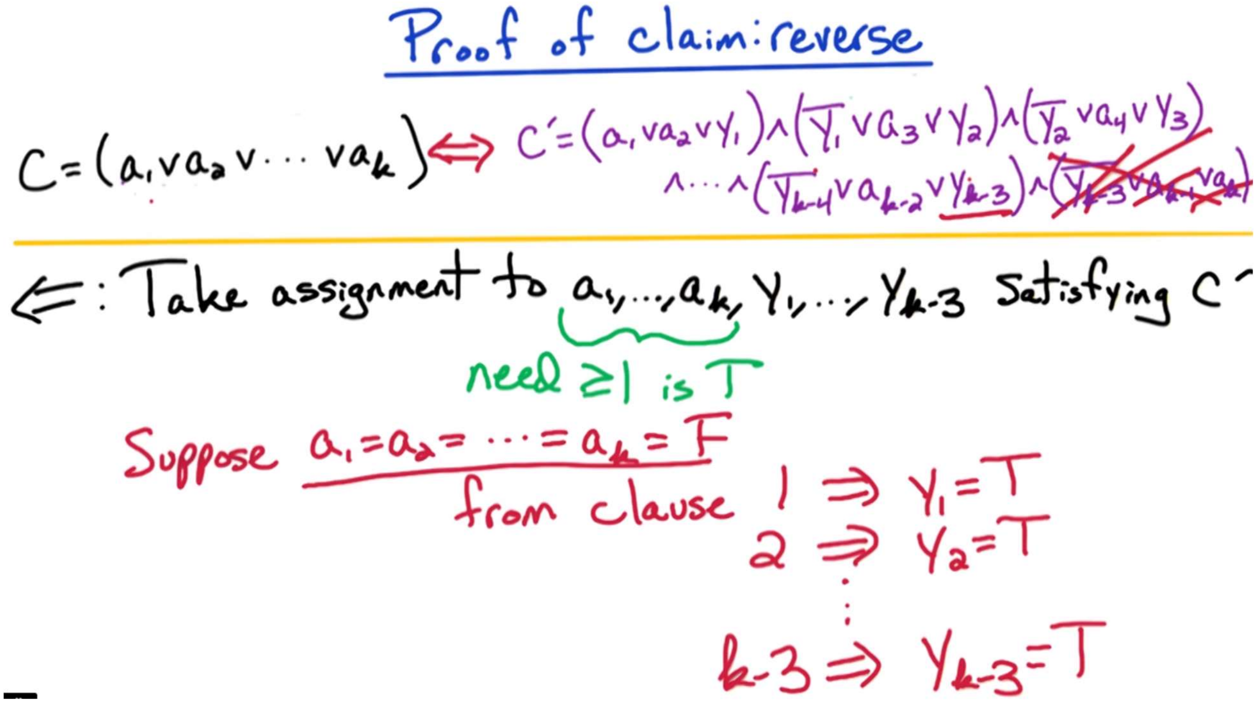
Let's suppose i = 4. So, we have a4 = True and this clause ($\overline{y2}$ ∨ a4 ∨ y3) is satisfied, so we can remove this clause. Now, what about the i-2 earlier clauses? Well, we can use these positive forms of these auxiliary variables to satisfy these earlier clauses. So, we set y1 and y2 to be true in this case and, in general, we set y1 thru y_{i-2} to True and this satisfies the first i-2 clauses of C′.

So, this setting of the first i - 2 auxiliary variables satisfies the first i - 2 clauses. ai set to True satisfies the (i–1)st clause. What do we do about the later clauses? Well, here we're going to use the negative form of these auxiliary variables. We set the remaining auxiliary variables to False. In this case, $\bar{y}_{k-4}$ and $\bar{y}_{k-3}$ and in general we set y_{i-1} through y_{k-2} to false and this satisfies the remaining clauses which appear after the (i-1)st clause.

The punchline is that this literal (ai = T) satisfies the (i-1)st of C′. We use these auxiliary variables (y1 thru y_{i-2}) to satisfy the earlier clauses and we use these auxiliary variables (y_{i-1} thru y_{k-2}) to satisfy the later clauses. So, we only need one literal of C to be satisfied, and then we can use the auxiliary variables to satisfy all the other clauses of C′. Now, let's do the reverse implication.

***

(Slide 12) General Claim: Reverse



Proof of claim: reverse

$C=(a_1 \vee a_2 \vee \cdots \vee a_k) \Leftrightarrow C'=(a_1 \vee a_2 \vee y_1) \wedge (\overline{y_1} \vee a_3 \vee y_2) \wedge (\overline{y_2} \vee a_4 \vee y_3)$
$\wedge \cdots \wedge (\overline{y_{k-4}} \vee a_{k-2} \vee \underline{y_{k-3}}) \wedge \cancel{(\overline{y_{k-3}} \vee a_{k-1} \vee a_k)}$

$\Leftarrow$: Take assignment to $\underbrace{a_1,\ldots,a_k}_{\text{need } \geq 1 \text{ is T}}, y_1,\ldots,y_{k-3}$ Satisfying C'

Suppose $\underline{a_1=a_2=\cdots=a_k=F}$

from clause $1 \Rightarrow y_1=T$

$2 \Rightarrow y_2=T$

$\vdots$

$k\text{-}3 \Rightarrow y_{k-3}=T$

Now let's prove the reverse implication.

C′ => C:

So let's take an assignment to these original k literals and these auxiliary k-3 variables which satisfy C′. Let's prove that there is a satisfying assignment for C.

What we'll do is we'll just ignore these auxiliary variables and we'll prove that these are settings of the original k literals that satisfy the original clause.

Now, in order to satisfy this clause C, we just need to show that at least one of these literals is set to true. Let's suppose that's not the case. Suppose all of these k literals are set to False. Under this assumption, is it possible to satisfy C′? We'll show it's not possible.

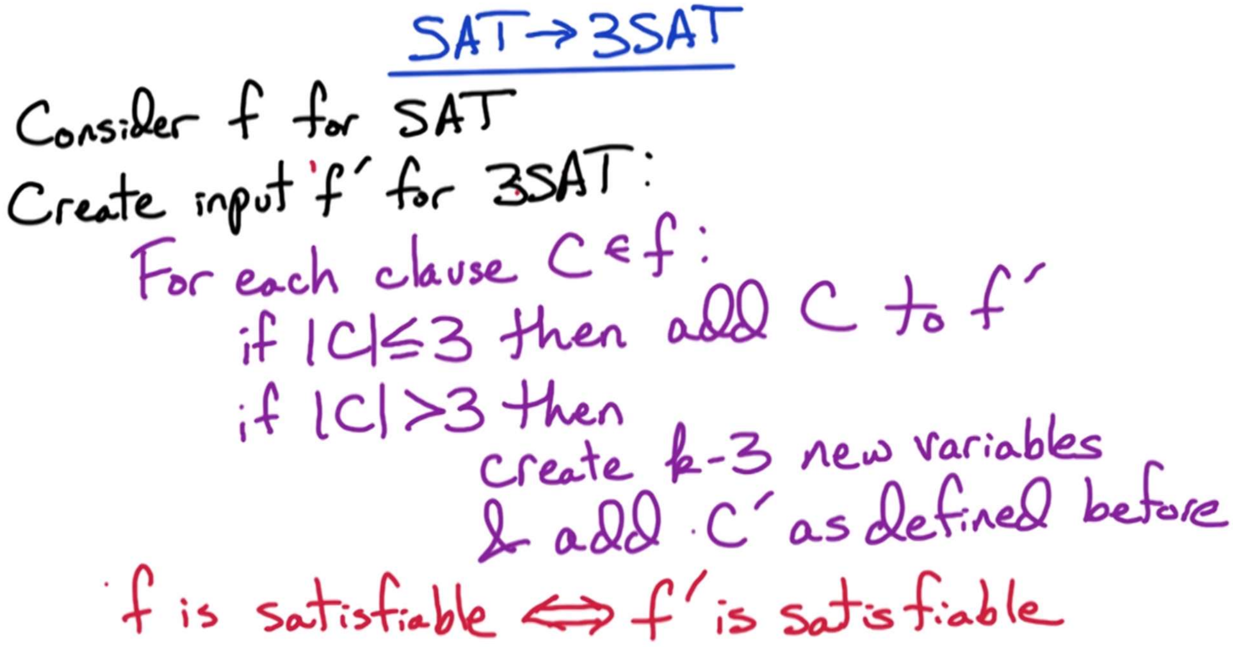
Now we're supposing we have an assignment which satisfies C′. So it satisfies all of these clauses. Let's look at the first clause. Now we're supposing that a1 and a2 are set to false. So the first two literals are not satisfied. So the third literal must be satisfied. That means y1 must be set to True. That's the only way to satisfy this clause, under this assumption. Similarly, let's look at the second clause. What we're seeing Y1 is true. So this literal is not satisfied. Also a3, we're assuming is set to false. So this literal is not satisfied. So, we better satisfy this third literal, y2. So, y2 has to be set to True.

Continuing on, look at the penultimate clause. In order to satisfy this penultimate clause, we have to satisfy this literal - we have to set y_{k-3} to be True. Now look at the last clause, where y_{k-3} is set to True. So this literal is not satisfied. Similarly, these last two literals are not satisfied because they're set to False. So, this clause is not satisfied. That means that C prime is not satisfied. That's a contradiction.

We were assuming that this was assignment which satisfied C′. So that means that this assumption that all of these literals are set to False is not true. So at least one of them must be set to True and therefore that literal satisfies this clause C. So if we just ignore the assignment to these auxiliary variables, then the setting for the original literals satisfies this original clause.

***

(Slide 13) SAT 3SAT



SAT → 3SAT

Consider f for SAT

Create input f′ for 3SAT:

```
    For each clause $C \in f$:
        if $|C| \leq 3$ then add C to f′
        if $|C| > 3$ then
            create k-3 new variables
            & add C′ as defined before
f is satisfiable $\Leftrightarrow$ f′ is satisfiable
```

Now we can formalize our reduction from SAT to 3SAT.

So let's consider our input formula for the satisfiability problem. And, let's create an input formula for the 3SAT problem, by the following procedure:

> Let's consider the clauses of f, one by one. So, for clause C in f, we'll have two cases depending on whether the clauses of size are most three, or strictly bigger than three.
>
> 1. If C contains at most three literals, then we can add this clause as is to this new formula.
> 2. Now what if C contains more than three literals? Then we have to use our previous construction. In this case, we create k-3 new variables, as we said before, and we replace C by this new formula C′ - this sequence of k-2 clauses, as we defined before.

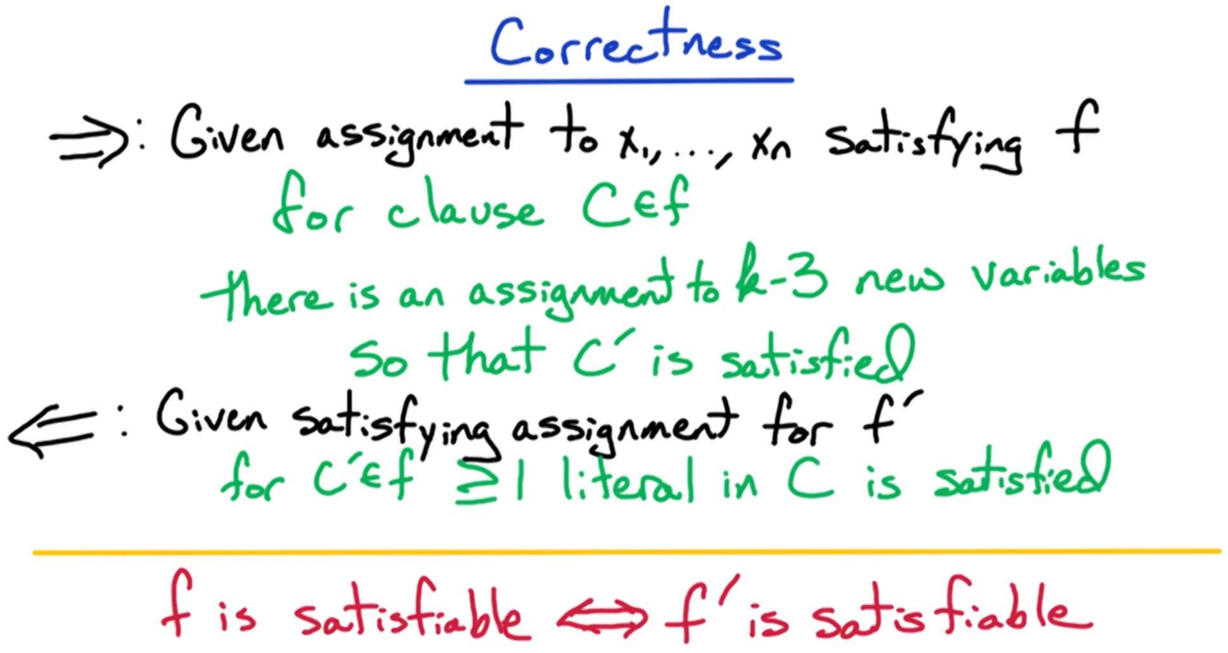
So, small clauses stay the same; big clauses are replaced by k-2 new clauses, as we defined before.

Now that we've defined the input to the 3SAT problem, now we have to prove that this original input to the SAT problem is satisfiable, f is satisfiable if and only if its input to the 3SAT formula f′, is satisfiable. So f is satisfiable if and only if f′ is satisfiable. Now this statement is quite straightforward to prove, given that we've already proven that C is satisfiable if and only if C′ is satisfiable.

Then afterwards, the remaining task is to show that given a satisfying assignment to f′, we can construct a satisfying assignment to f. That again will be straightforward, because we just ignore these auxiliary variables and the setting for the original variables will give us a satisfying assignment to the original formula.

***

(Slide 14) 3SAT Correctness



Correctness

$\Rightarrow$: Given assignment to $x_1, \ldots, x_n$ satisfying f

for clause $C \in f$

there is an assignment to k-3 new variables

so that C' is satisfied

$\Leftarrow$: Given satisfying assignment for f'

for $C' \in f'$ ≥ 1 literal in C is satisfied

f is satisfiable $\Leftrightarrow$ f' is satisfiable

Let's prove this equivalence and hopefully it seems obvious given what we've shown so far.

So, let's start with the forward implication.   So let's take an assignment to the original n variables which satisfies the original formula f. Now we want to show that keeping this assignment the same for these original n variables, there is an assignment to the new variables so that this new formula f′ is satisfied. Now, since we're keeping the assignment for these n variables the same and the new variables are distinct for each new clause. Therefore, we can look at it a clause by clause.

So, let's look at clause C in the original formula f. If C has at most three literals then it's obvious. If C is bigger than size three then we replace C by this sequence of k-2 clauses, called C′. Now, the key is that C′ uses k-3 new variables. These variables only appear in C′ for C - they don't appear in any other sequence of clauses.   So the new variables for C are distinct from the new variables for any other clause. So we can set these variables however you want with respect to C and this won't affect any other clauses.

And what we saw before is that there is an assignment to these k-3 new variables so that this new formulas C′ is satisfied. In particular, we took the first literal satisfied in C, call it ai and then we set the first i - 2 variables to True, and the remaining new variables to False, and we showed that that satisfied C′.

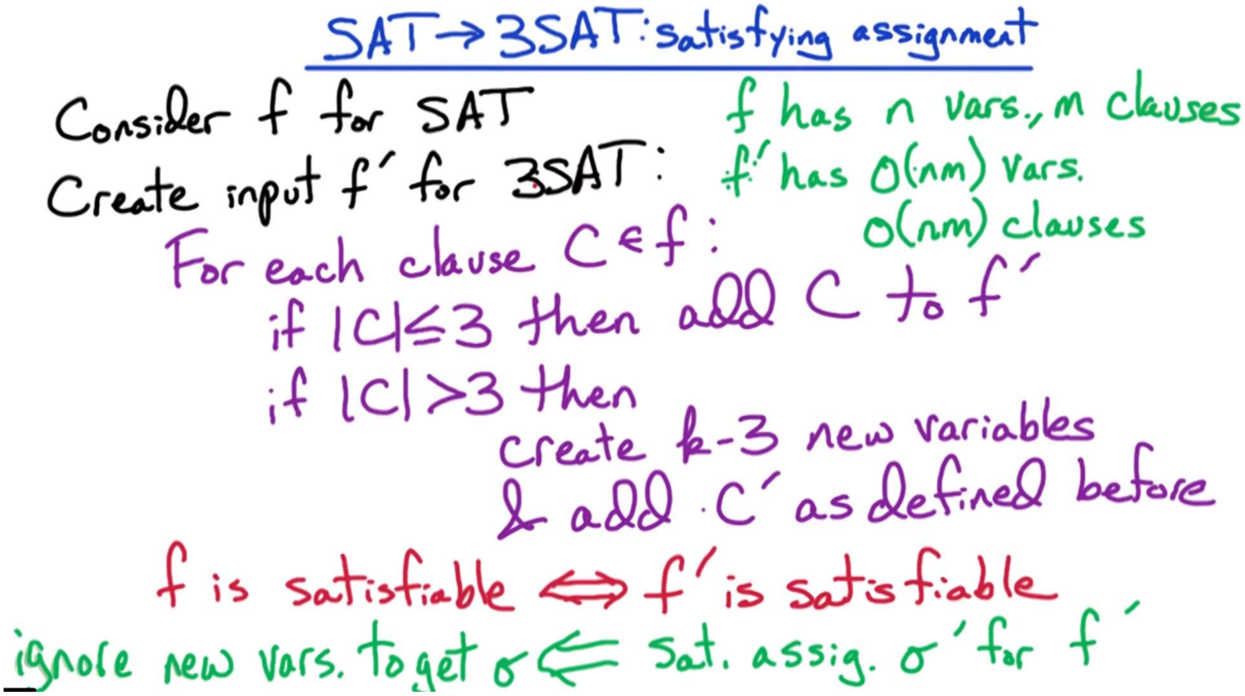
Now let's look at the reverse implication. Let's take a satisfying assignment for f′. And we want to ignore the assignment for the new variables and show that the assignment for the original n variable satisfies f.

Consider a sequence of clauses corresponding to some C′ ∈ f. What we saw before is that at least one of the literals in the original clause C must be satisfied. If all of these literals in the original clause were set to False, then, there would be no way to satisfy this sequence of clauses C′.

So therefore we can ignore the setting on the new variables and just looking at the setting on the original n variables - That's going to satisfy each of the clauses in the original formula f.

***

(Slide 15) Satisfying Assignment



SAT → 3SAT: satisfying assignment

Consider f for SAT — f has n vars., m clauses

Create input f′ for 3SAT: — f′ has O(nm) vars. O(nm) clauses

```
For each clause $C \in f$:
    if $|C| \le 3$ then add C to f′
    if $|C| > 3$ then
        create k-3 new variables
        & add C′ as defined before
```

f is satisfiable ⇔ f′ is satisfiable

ignore new vars. to get $\sigma \Leftarrow$ Sat. assig. σ' for f′

Now going back to our earlier reduction, we showed how to take input formula for SAT and transform it into an input formula for 3SAT. This was the reduction for creating f′. And then we just proved that the original formula f is satisfiable if and only if this new formula, f′, is satisfiable.

Now, what remains? While we run our 3SAT algorithm, our blackbox algorithm on f′, if it produces NO, that there is no satisfying assignment, then we say NO, there is no satisfying assignment for f. What if it produces a satisfying assignment? So, suppose it gives us a satisfying assignments Σ′ which satisfies f′. Well, we have to produce a satisfying assignment for f. How do we transform this satisfying assignment, Σ′, to a satisfying assignment for f?

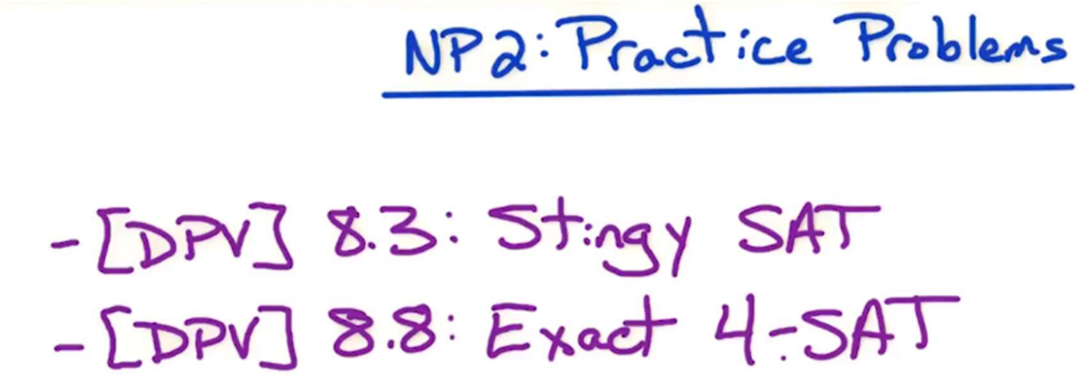
What we just saw from this proof is that if we ignore the assignment for the new variables and keep the assignment for the original variables the same, then we get a satisfying assignment for f. So we take this satisfying assignment for f′, we ignore the assignment for all of the new variables and the assignment for the original variables gives us a satisfying assignment for f.

And that completes our reduction. One last thing to note is what is the size of f′? f, our original input to SAT, has n variables and m clauses. How many variables does f′ have in the worst case? Well, we might create n new variables for each clause. So, it has O(nm) variables in the worst case. And we're also replacing every clause by order n clauses, in the worst case. So we have O(nm) clauses. But, this is okay because the size of f’ is polynomial in the size of f. So we have an algorithm which is polynomial running time in the size of f’. It's still polynomial in the size of f as well.

So this completes our first NP-completeness proof.

***

(Slide 16) Practice Problems



NP2: Practice Problems

-[DPV] 8.3: Stingy SAT

-[DPV] 8.8: Exact 4-SAT

Now that you've seen your first NP-completeness proof - the fact that 3SAT is NP-complete, there are a few relevant practice problems from the text book that you can try now.

- In problem 8.3, they consider a variant of SAT, called Stingy SAT, and you want to prove that stingy SAT is NP-complete.
- In problem 8.8, you consider Exact 4-SAT. This means that every class has exactly four literals, not at most four literals, but exactly four literals in every class. And you want to prove that Exact 4-SAT is NP-complete.