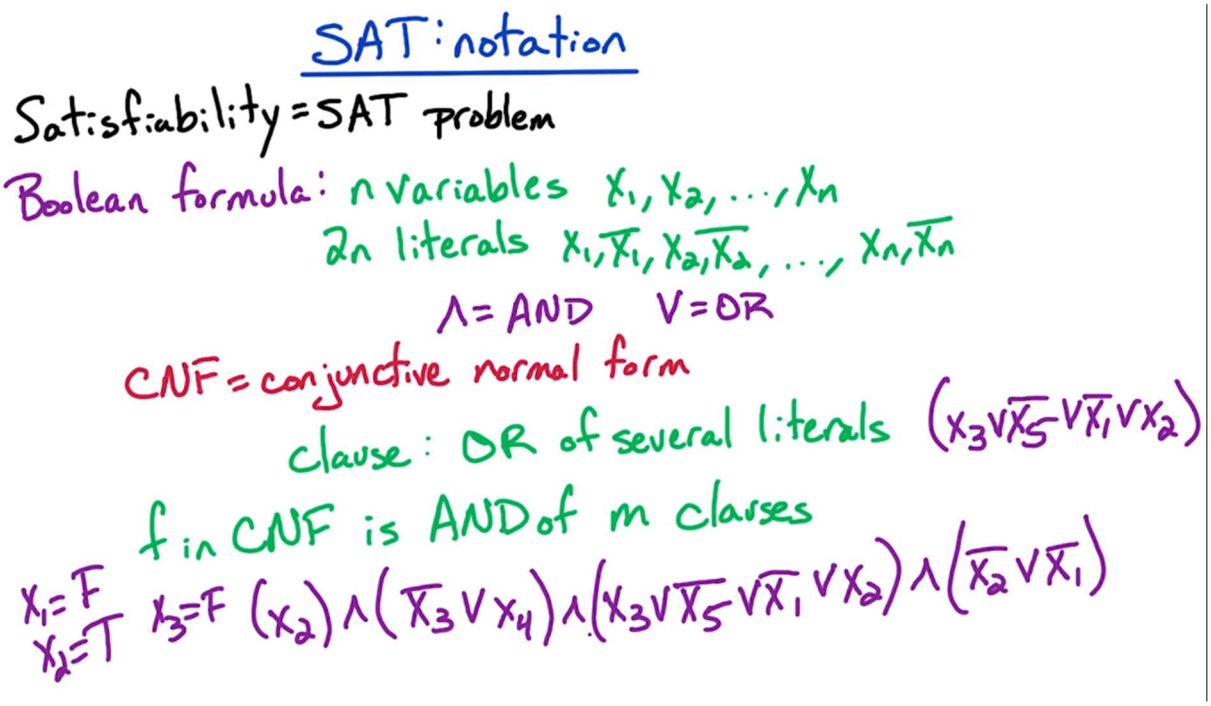
LESSON: GR2: 2-Satisfiability

***

(Slide 1) SAT Notation




Let's look now at an application of our strongly connected components algorithm. We're going to look at the satisfiability problem, also known as the SAT problem. This problem is going to play a central role later in our study of NP completeness.

Let's start with some terminology:

- We're going to be looking at **boolean** formulas.
- We're going to have n **variables,** x1, x2 up to xn.
- It's a boolean formula so these variables are going to take **True** or **False**.
- There are 2n **literals**. These refer to the positive and negative forms of the variables x1 and $\overline{x1}$ (you say "x1 bar"), x2 and $\overline{x2}$ (you can also say "x2 complement") and so on up to xn and $\overline{xn}$ ("xn complement").
- Our formulas are going to be composed of NOTs, ANDs and ORs. And we'll use the notation this symbol (∧ = AND symbol) for the logical AND, and this symbol for the logical OR (∨ = OR symbol). Personally, to help myself remember, notice that this symbol ∧ looks kind of like an A.
- Now we're going to look at formulas in **CNF**. This stands for **Conjunctive Normal Form**. Let me tell you what that is:
    - The formula would be composed of several **clauses**.

- Each **clause** is the OR of several literals, for example: $(x3 \vee \overline{x5} \vee \overline{x1} \vee x2)$ - that's a **clause**. In order to satisfy that clause, I have set x3 to be True, x5 to be False, x1 to be false, or x2 to be true.
- Finally to construct a formula in CNF, we're going to take the AND ($\wedge$) of m clauses.
- Here's an example of a formula in CNF:

$$(x2) \wedge (\overline{x3} \vee x4) \wedge (x3 \vee \overline{x5} \vee \overline{x1} \vee x2) \wedge (\overline{x2} \vee \overline{x1})$$

It has four clauses, each clause is the OR of some literals. Some are of size one. So, there's a unit clause. There's a clause of size two, clause of size four, clause of size two. And then we take the AND of these clauses.

So in order to satisfy this formula, we have to satisfy at least one literal in every clause. For example, for this formula, if I set x1 to be false, x2 to be true and x3 to be false, then it doesn't matter the setting for x4 and x5, I satisfy this formula.

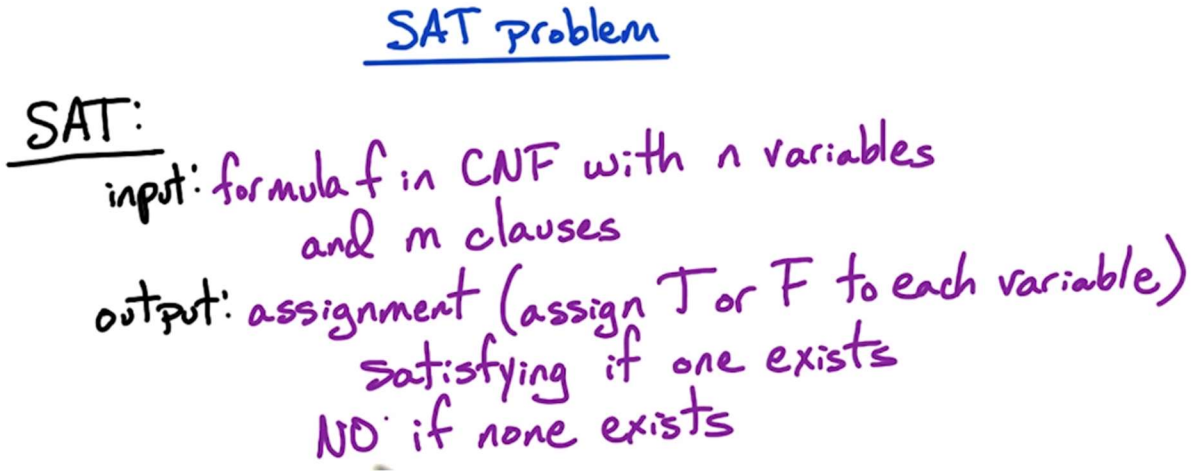
Plug in x1 to be false, that satisfies this clause, x1 to be false satisfies this clause as well, x2 to be true satisfies this clause, and x3 to be false satisfies this clause. So all the clauses are satisfied, so the end of these clauses is satisfied. So, the formula itself is satisfied. So this formula evaluates to True for this assignment of the True/False to the variables.

- Now any formula can be converted into a CNF form, but the size of the formula may blow up.

Now, let's go ahead and precisely define the SAT problem.

***

(Slide 2) SAT Problem



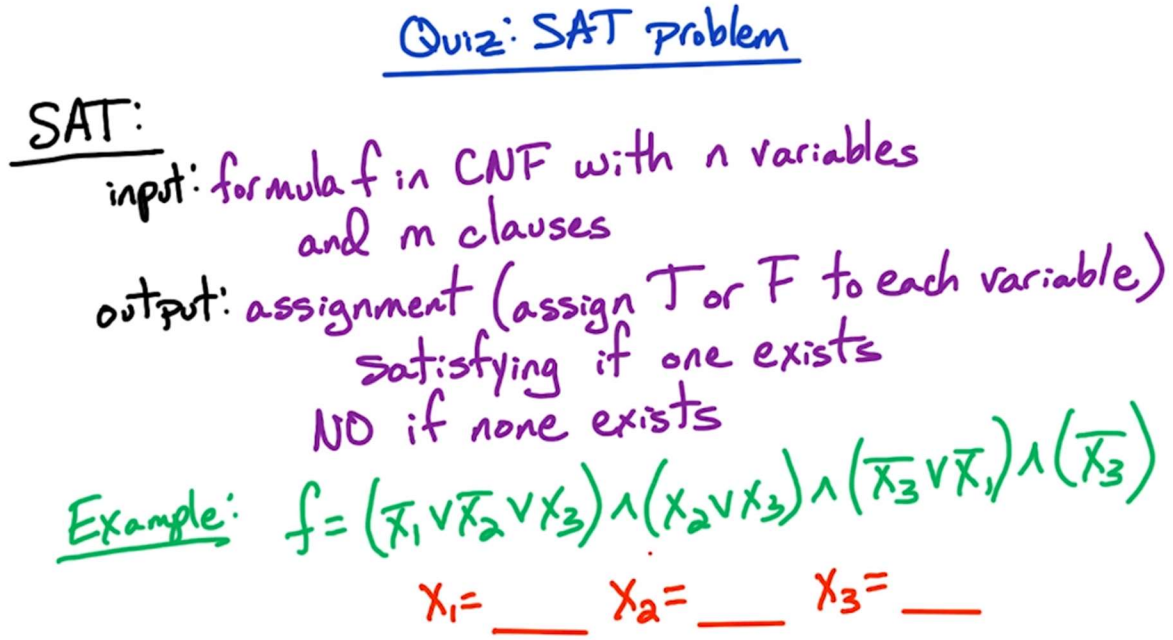

The input to the SAT problem is a formula f in Conjunctive Normal Form, which is composed of n variables; x1 through xn, and m clauses.

The output is a satisfying assignment if one exists. This is an assignment of True or False to each of the n variables so that the formula evaluates to True.

Now if there is no satisfying assignment, no assignment where the formula evaluates True, then we output NO.

***

(Slide 3) SAT Problem Quiz

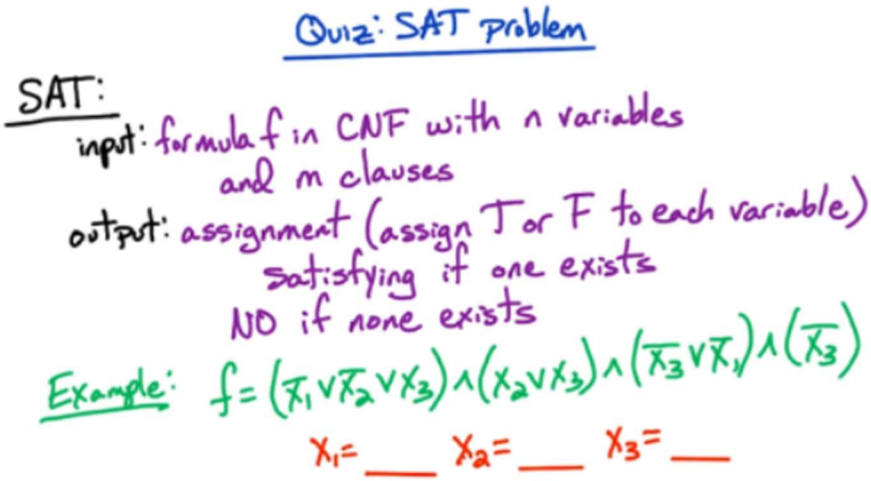



Now here's an example of a SAT input: The formula f is:

$$f = (\overline{x1} \vee \overline{x2} \vee x3) \wedge (x2 \vee x3) \wedge (\overline{x3} \vee \overline{x1}) \wedge (\overline{x3})$$

To make sure you understand this problem, why don't you go ahead and specify the output for this input? Either give a satisfying assignment if one exists, or NO, otherwise.

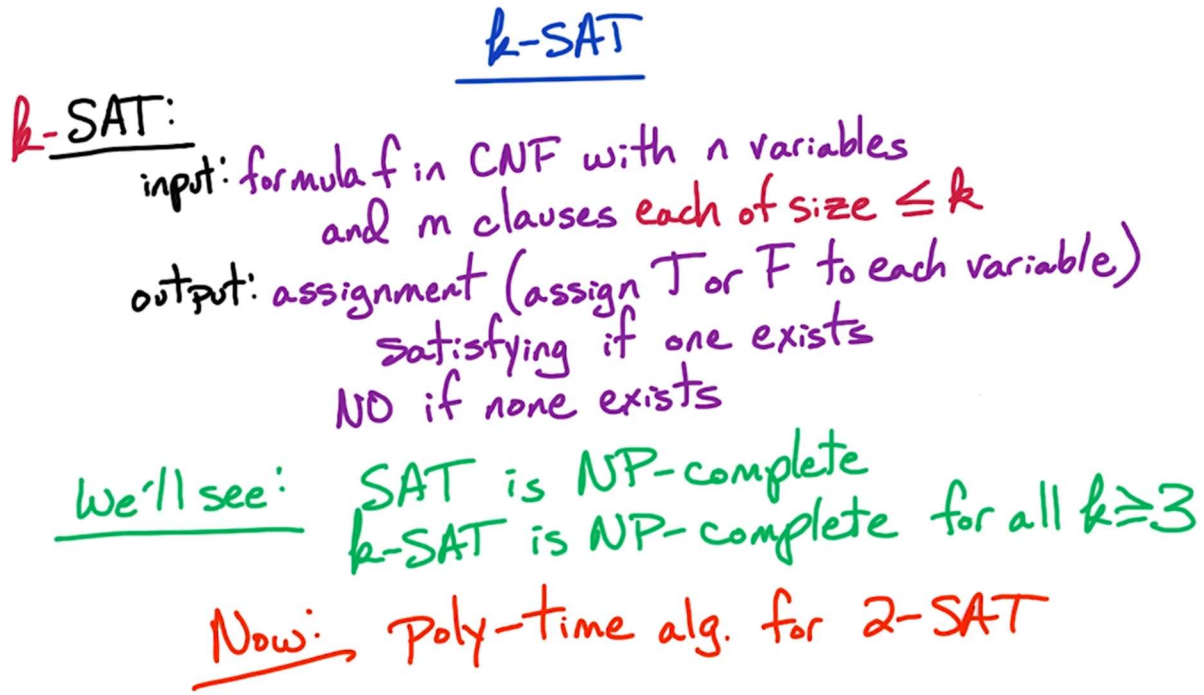
GR2: 2-Satisfiability: SAT Problem Quiz



Specify a satisfying assignment for the above formula f, if one exists, and NO, otherwise. For each value, choose if it will be assigned True or False

***

(Slide 4) k-SAT




A satisfying assignment for this formula is the assignment which sets x1 to be False, x2 to be True, x3 to be False.

Now, we're going to look at a restrictive form of the SAT problem called k-SAT. For example, for 3-SAT, the input is going to be a formula in CNF with n variables and m clauses. But now, all the clauses are of size, at most, 3, or k in general. The size of a clause is the number of literals in it.

So, this formula is an example of a 3-SAT formula. Actually, it's an example of a k-SAT formula for all k at least 3.

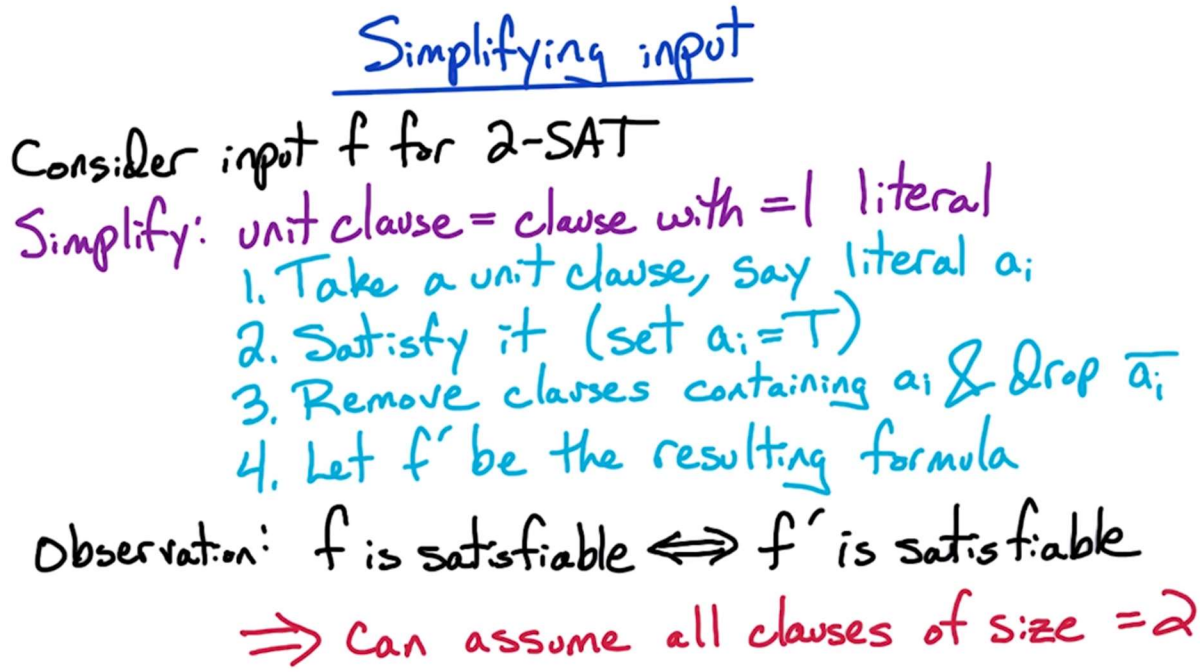
Now, what we're going to see later is that SAT problem is NP-complete. And for every k at least 3, we'll see that the k-SAT problem is NP-complete. So, 3-SAT is NP-complete, 4-SAT is NP-complete, and so on. What we're going to see now is a polynomial time algorithm for 2-SAT.

So, there's a very interesting dichotomy for 2-SAT - that there's a polynomial time algorithm and we're going to solve it using our strongly connected components algorithm. And then, when k is at least 3, 3-SAT is NP-complete.

So let's dive into our algorithm for 2-SAT.

***

(Slide 5) Simplifying Input




Consider input f for 2-SAT problem. Here's an example input for 2-SAT:

$$(x3 \vee \overline{x2}) \wedge (\overline{x1}) \wedge (x1 \vee x4) \wedge (\overline{x4} \vee x2) \wedge (\overline{x3} \vee x4)$$

This formula consists of 4 variables x1, x2, x3, x4 and it has five clauses.

Now I want to simplify the input to our 2-SAT problem. In particular, I want to remove **unit clauses**. What's a unit clause? This is an example of a unit clause right here: $(\overline{x1})$ - this is a clause with exactly one literal in it.

Now how can I remove it? Well, in order to satisfy this formula, I know that I have to satisfy this clause. In order to satisfy this clause, there's only one way: I have to set x1 to be False. So, I might as well go ahead and set x1 to be False and the resulting formula will be True satisfiable if and only if the original formula was satisfiable.

So, here's the basic procedure for eliminating unit clauses:

1. Take any unit clause, in this case $(\overline{x1})$, and let's call that literal a_i. So in this case a_i is $(\overline{x1})$.

2. Then I'm going to satisfy that literal. So I'm going to set a_i to be True.

   In this case, I'm going to set $\overline{x1}$ to be True. What does it mean to set $(\overline{x1})$ to be True? That means to set x1 to be False. If I set the variable x1 to be False, then I satisfy this literal.

3. Now if I set x1 to be False, then any clause which contains $\overline{x1}$, can be eliminated because those clauses are satisfied. There are no other clauses containing $\overline{x1}$ , so I can't eliminate any other clauses in this example.

   Now what else can I do? What about this x1 in (x1 ∨ x4)? Well, I know x1 is set to False, so this literal will not be satisfied. So I might as well remove it. So this clause now becomes just x4 by itself. So I drop all occurrences of $\bar{a}i$ , then this formula then reduces to $(x3 \vee \overline{x2}) \wedge (x4) \wedge (\overline{x4} \vee x2) \wedge (\overline{x3} \vee x4)$.

4. The third clause got reduced and the second clause got eliminated. Let's call the resulting formula $f'$. So this simplified formula is $f' = (x3 \vee \overline{x2}) \wedge (x4) \wedge (\overline{x4} \vee x2) \wedge (\overline{x3} \vee x4)$. The original formula is f.

   What is the claim? - the claim is that f, the original formula is satisfiable if and only if this reduced formula f' is satisfiable.

   Then what can I do?

5. Well now I can take this simplified formula f', and again there's another unit clause which got formed and now I can set x4 = True, and then I can simplify the formula and I can keep going. Eventually I either end up with the empty formula, which is satisfied, or I end up with a formula where all clauses are of size exactly two.

   And that's how I'm going to simplify my input for 2-SAT problem. So I can take an arbitrary input for the 2-SAT problem and I can simplify it so that all of the clauses are of size exactly two by repeating this procedure over and over.

Now the key observation that we just mentioned is that the original formula f is satisfiable if and only if this reduced formula f' is satisfiable. This is clearly true since the only way to satisfy f, the original formula, is to take this unit clause, satisfy it (and therefore I must satisfy this literal). And then steps 2 and 3 are forced.

Now, the implication of this procedure is that I can keep applying it over and over as long as I have unit clauses and eventually the formula is either satisfied or all clauses are of size exactly

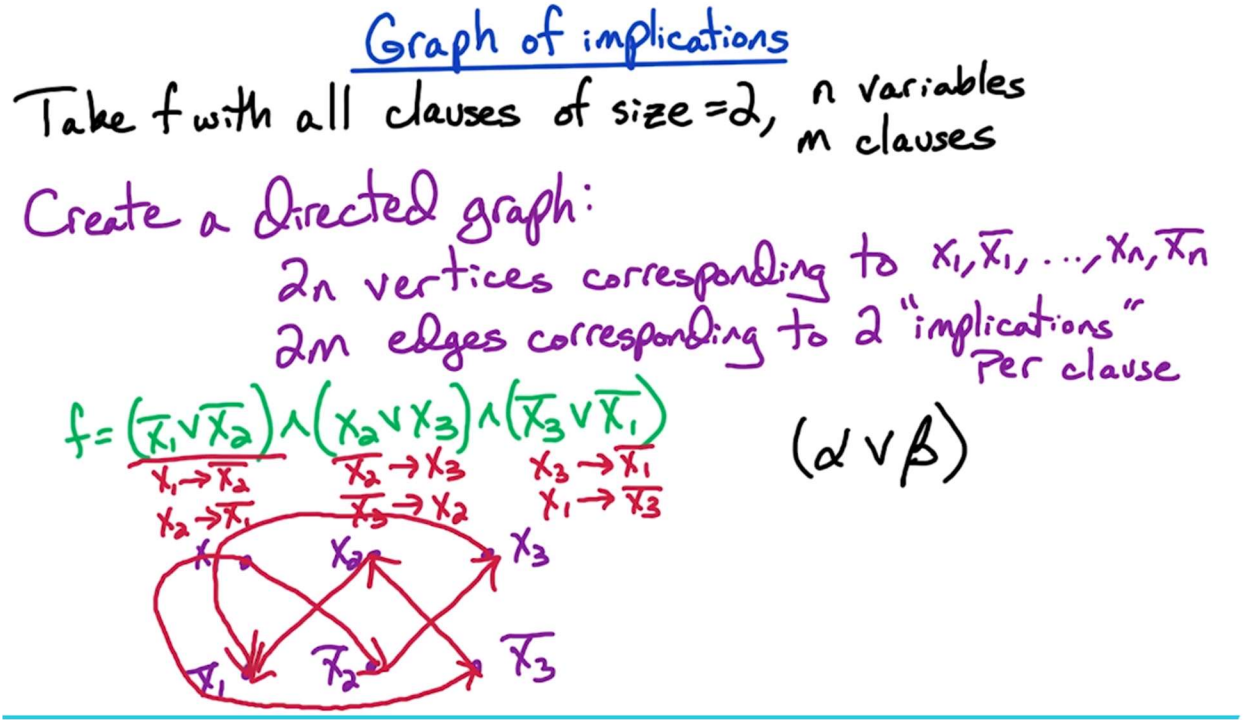
two.

So now in order to design an algorithm, I can assume that the input to my 2-SAT problem - all of the clauses in that input - are of size exactly two. This is going to make the input to the problem slightly easier for us.

Now let's go ahead and see how this 2-SAT problem relates to the strongly connected components algorithm.

***

(Slide 6) Graph of Implications




Let's take our input to the 2-SAT problem. This is the formula, f. And we can assume that all clauses are of size exactly two, as we just mentioned. And let's let n denote the number of variables in our formula. Let's call them x1 through xn, and let's say m is the number of clauses in our formula.

We want to convert this logic problem into a graph problem. So, what we are going to do is we're going to take this Boolean formula and we're going to create a directed graph. This graph is going to encode all of the information in this formula.

Now, first off, the vertices of this graph are going to correspond to the variables of this formula. There are n variables and there are 2n literals - x1, $\overline{x1}$, x2, $\overline{x2}$, … , xn, $\overline{xn}$.. So we're going to have 2n vertices. One vertex for each literal, and then we're going to have 2m edges in this directed graph. We're going to have two edges for each clause.

Each clause has two implications, and the two edges are going to correspond to the two implications per clause. Let's look at a specific example and then we can see what we mean by implications per clause.

Here's a sample formula with three variables and three clauses:

$$f = (\overline{x1} \wedge \overline{x2}) \wedge (x2 \wedge x3) \wedge (\overline{x3} \wedge \overline{x1})$$

Our graph is going to have six vertices corresponding to the six literals, x1, $\overline{x1}$, x2, $\overline{x2}$, x3, $\overline{x3}$.

And let's look at the first clause, $(\overline{x1} \wedge \overline{x2})$. Suppose that we set x1 to be True. So this first literal is not satisfied, then we better satisfy the second literal in order to satisfy the formula. So if x1 is set to True, then what does that mean about x2? Then x2 better be set to False in order to satisfy this formula.

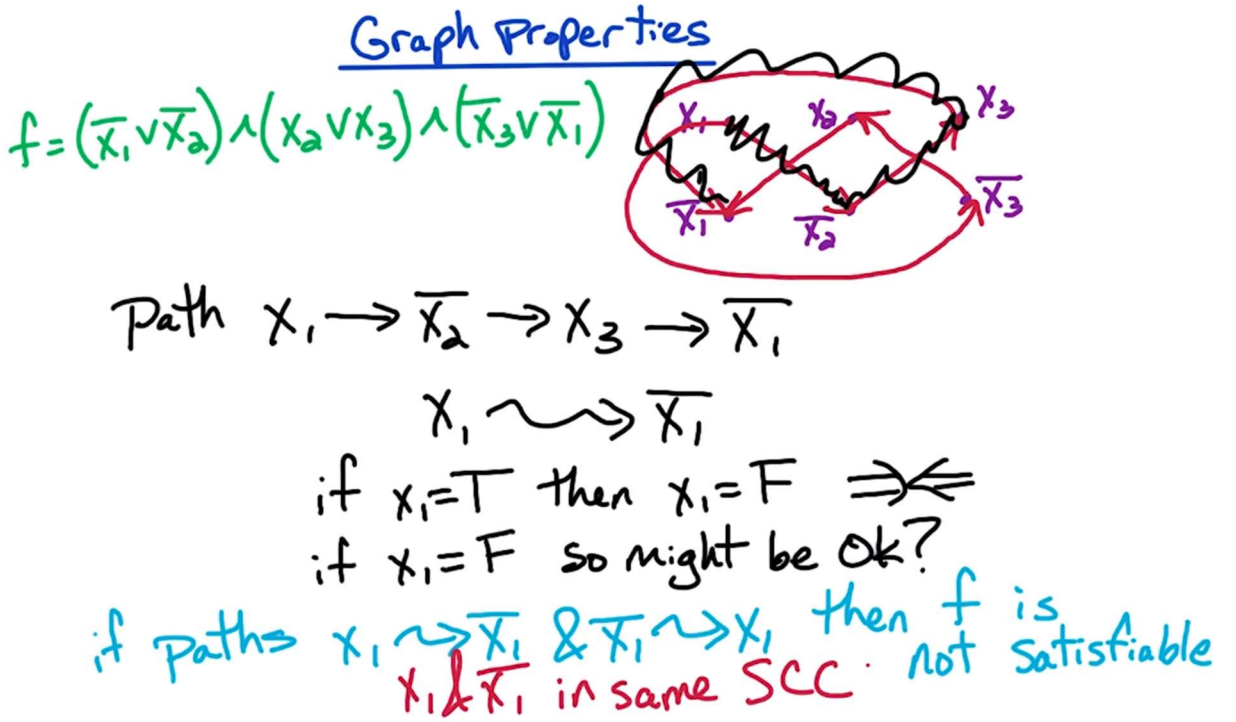
Similarly if x2 is set to True. So this literal is not satisfied, then we've got to satisfy the first literal. So we need to set x1 to be False.

So to encode these implications, we're going to add in the edge from x1 to $\overline{x2}$. If this literal x1 is satisfied so x1 is set to True then we have to satisfy the literal $\overline{x2}$. So we have an edge from x1 to $\overline{x2}$. And similarly, we have an edge from x2 to $\overline{x1}$. Similarly, for this clause we have an edge from $\overline{x2}$ to x3 and from $\overline{x3}$ to x2. Finally, for the last clause we have an implication from x3 to $\overline{x1}$ and from x1 to $\overline{x3}$.

So this is our graph corresponding to this formula. And, in general, if we have a clause which has literals $\alpha$ and $\beta$, so we have $(\alpha \vee \beta)$ is the clause, then we're going to have the edges from $\bar{\alpha}$ to $\beta$ because if we don't satisfy $\alpha$ then we have to satisfy $\beta$ and we are going to have the other edge from $\bar{\beta}$ to $\alpha$.

***

(Slide 7) Graph Properties




Now, here's our specific 2-SAT example from earlier, and here's the graph corresponding to this Boolean formula. Let's take a look at this graph and explore some properties of it.

Here's a particular path which is quite interesting. We go from x1, we follow this edge to $\overline{x2}$, to x3, and then we can follow it to $\overline{x1}$. So, we have this path which goes from x1 and ends up at $\overline{x1}$.

Now this is a path of implications. Each edge is an implication. So, if we follow those implications out - so if we set c1 to be true, we satisfy this literal, then we're going to have to satisfy this literal. How do we satisfy this literal? By setting x1 to be False. (if we had set x1 to be True, then in order to satisfy the formula, we have to set x1 to be False. That's clearly a contradiction. So, there's no way we can satisfy the formula by setting x1 to be True).

Now, what happens if we set x1 to be false? Well, there are no edges out of $\overline{x1}$. So, there are no implications from setting x1 to be false. So, it might be okay. We don't know. We have to proceed with the other variables. All we know is that, because there is a path from x1 to $\overline{x1}$, clearly we cannot set x1 to be true.

Now, what if there was a path from x1 -> $\overline{x1}$ and from $\overline{x1}$ -> x1? Then we know that we can't

set x1 to be true, because that would lead to a contradiction, and we can't set x1 to be false, because that would lead to a contradiction. So, there's no valid setting of x1. There's no setting of x1 which leads to a satisfying assignment. Therefore, we can conclude that f is not satisfiable.

Now, we saw that if there is a path from x1 to $\overline{x1}$ , then we can't set x1 to be true. We cannot satisfy the formula by setting x1 to be true. Similarly, suppose there's a path from $\overline{x1}$ to x1, then we know that we can't set x1 to be false. So now, suppose that there is a path from x1 to $\overline{x1}$ and also there's a path from $\overline{x1}$ to x1, then what do we know about f?

Well, we know we can't set x1 to be true and we know that we can't set x1 to be false. So in fact, there is no way to satisfy the formula f. What does this mean about the graph?

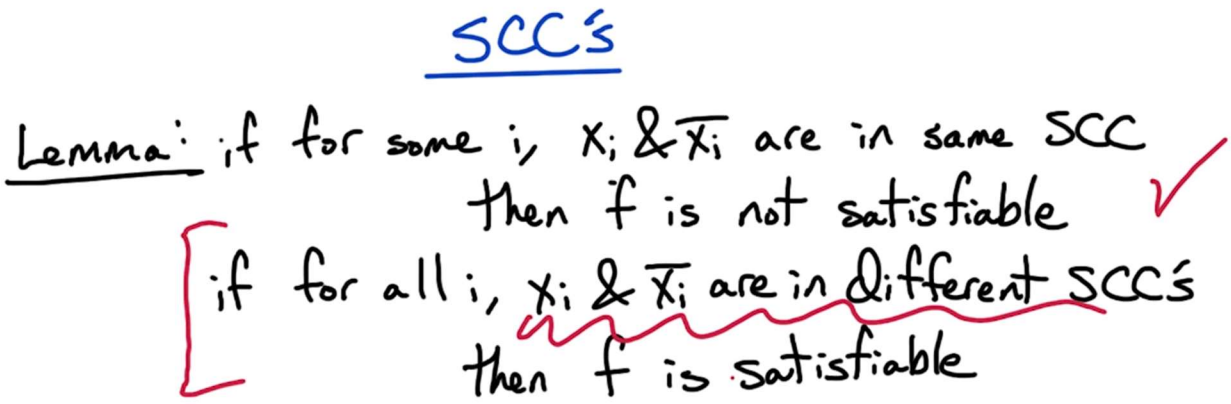
If there is a path from x1 to $\overline{x1}$ and from $\overline{x1}$ to x1, that means that these two vertices are in the same strongly connected component. They're strongly connected with each other, so they're in the same SCC. So, if this literal and its negation are in the same SCC, then this formula is not satisfiable.

And if this is true for any variable, the positive literal x_i and $\overline{x\_i}$ are in the same SCC, then the formula is not satisfiable. What we're going to see is if this is not true.

So, if, for every variable, the literal and its negation are in different SCCs - so, they never lie in the same SCC. Then, in fact, we're going to find a satisfying assignment. So, we're going to prove it satisfiable by finding a satisfying assignment.

***

(Slide 8) SCC's




Now let's formalize what we've just said. If, for some i, xi the positive literal and the negative literal $\bar{x}i$ are both in the same SCC, so these two literals are strongly connected with each other, then the formula is not satisfiable.

We just solved why this is true, because if we set xi to be true, then we get a contradiction because we have to set xi to be false. And, if we set xi to be false, then we get a contradiction because that leads to xi being true.

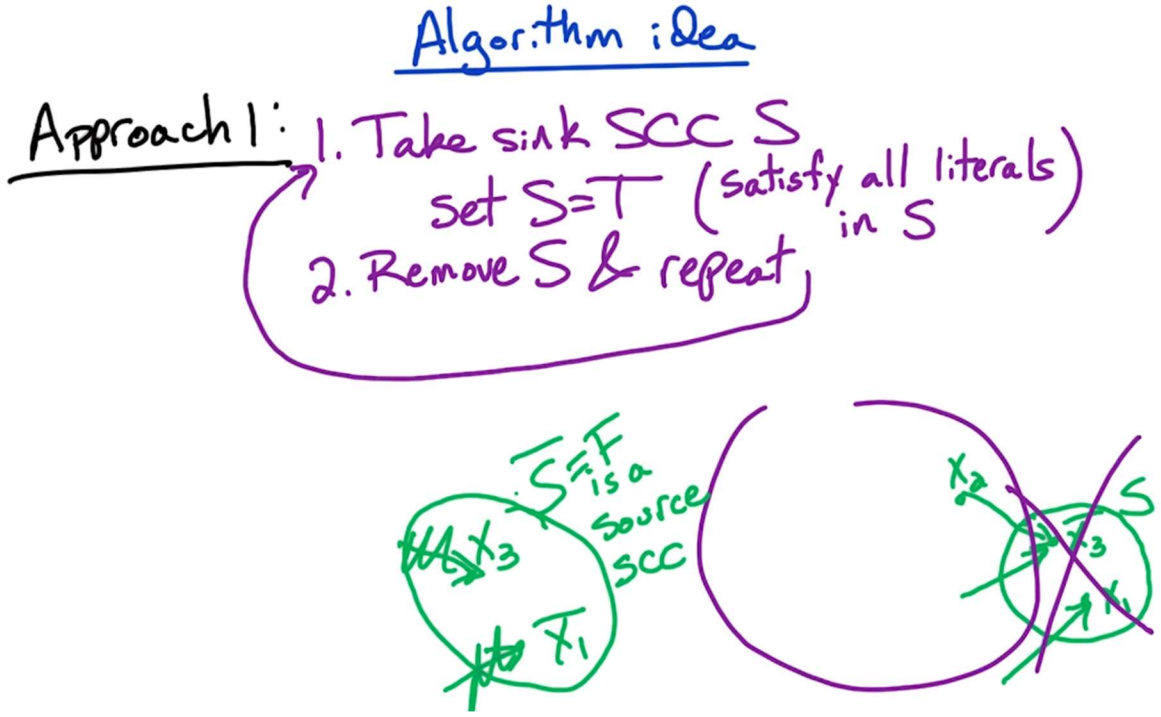
Now, suppose this is not the case: For every i, xi and xi bar are in different SCCs. In this case, f is satisfiable. So for every i, xi and $\bar{x}i$ are in different strongly connected components, then f is satisfiable. We'll prove this part by finding a satisfying assignment.

So this statement that if xi and $\bar{x}i$ are in the same SCC then f is not satisfiable. We already discussed why that's true. For this one, where for every i, xi and xi bar are in different strongly connected components. we're going to prove that f is satisfiable by finding a satisfying assignment. So we're going to construct an algorithm which is going to find a satisfying assignment and therefore we're going to prove that f is satisfiable by actually satisfying this f.

Let's go ahead and construct the algorithm. We're going to assume that for every i, xi and $\bar{x}i$ are in different strongly connected components. And we're going to use that fact to construct a satisfying assignment for f.

***

(Slide 9) Algorithm Idea




Now recall a strongly connected component algorithm. What was the main approach? We found a sink component. We marked those verses in the sink SCC and then we removed that sink SCC and recursed on the remainder graph.

What are we going to do here? We're going to do a similar approach. We're going to find a sink SCC. We're going to do an assignment for that sink SCC and then we're going to rip it out and work on the remainder graph.

So we're going to find a sink strongly connected component. Let's call it S.

As an example, here's a sink SCC. It contains $\overline{x3}$ and x1. They have edges coming in and no edge is going out. Now should we set these literals to be True or False? Well, we should set them to be True. Why? Well, let's consider this edge. Suppose the head of the edge is x2. Now if later in the algorithm we set x2 to be True, then this implication says that we have to set x3 to be False. So we have to satisfy this literal. So if we want to rip out this component and not worry about it again, then we better satisfy all these literals which means setting x3 to be False in this case and x1 to be True in this case.

So this is what we're going to do. We're going to go ahead and set this component to be True which means we're going to satisfy all the literals in S. So in this case we set $\overline{x3}$ bar to be

satisfied which means we set x3 to be False. And we want to satisfy x1, so we set x1 to be True.

Now there are no outgoing edges. So there are no implications that we have to follow because of this setting. There are in-coming edges but we satisfied all of these literals. Therefore, any incoming edges - the later assignment which may imply this implication - we don't have to worry about because we've already satisfied the tail of this implication.

So what can we do? We can rip out this component and work on the remainder of the graph. So we're going to remove this component and repeat the procedure on the remainder of the graph.

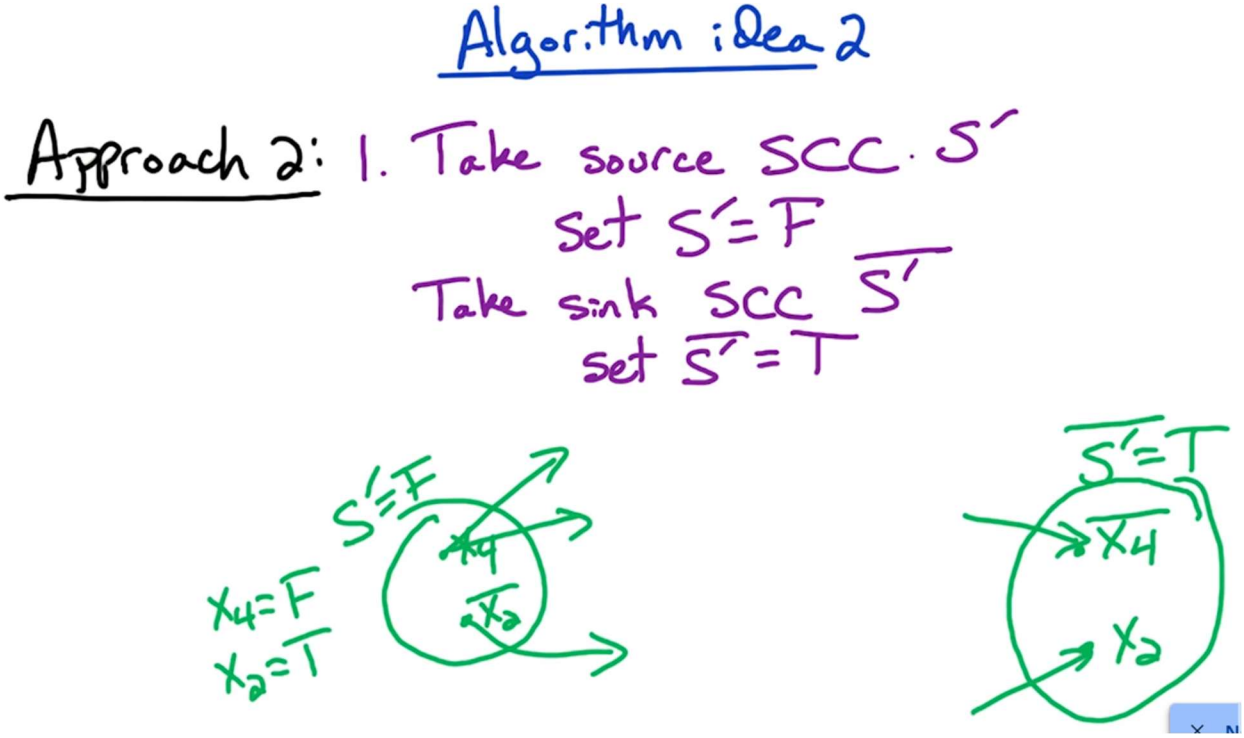
But, there's one problem. What about the complement of this set? What about x3 and $\bar{x}\imath$? By satisfying this literal $\overline{x3}$, we've not satisfied x3 and we've not satisfied $\bar{x}\imath$. So maybe there are edges in to x3 and $\bar{x}\imath$ and we have not satisfied these. And later we're going to have to follow this edge of implication and we're going to have to set x3 to be satisfied which means we're going to have a contradiction - we're going to have a problem.

Now what would be great is if this complement set $\bar{S}$ is a source SCC. If this set $\bar{S}$ is a source SCC then what do we know? Then we know it has no incoming edges. It's safe to set $\bar{S}$ to False because there is no incoming edges. And since we set it to False, we don't have to worry about any implications coming out of this set. It's a source SCC so their edge is coming out. But we don't have to follow those implications because we only have to follow those implications if the head of the edge is set to true. But the head of edge is set to False, so we don't have to follow the out edges and there are no in-coming edges. So it's safe to set it to False. So that's going to be the key.

If we take a sink SCC, then that sink we want to set to True. But then we have to worry about its complement. The key is that the complement set is going to be a source. So what do we want to do for a source? - we're going to set the source to be False.

***

(Slide 10) Algorithm Idea 2




Now, the previous idea we just discussed takes a sink SCC, and it satisfies that sink component. And then the issues come in by worrying about the complement of that set. Let's look at the reverse idea. Instead of taking a sink strongly connected component, let's take a source SCC. Let's call it S'.

So we have the source component containing x4 and $\overline{x2}$. Now, it's a source so it has no edges coming in, it only has edges coming out. Now, for a sink component, we wanted to set it to True.

What do we want to do for the source component? We want to set it to false. So by setting S' to False, we're going to NOT satisfy each of these literals. That means x4 is not going to be satisfied so x4 is set to false. $\overline{x2}$ is not satisfied. That means that x2 is set to true.

Now, these literals are not satisfied – but, there are no incoming edges so there are no later implications which are going to cause problems. Moreover, because these literals are not satisfied, we don't have to worry about the implications coming out of these literals, so we don't have to worry about these edges coming out. We can safely remove this component and work on the remainder of the graph.

But once again, what happens to the complement of this set? In this case, $\overline{x4}$ and x2? Well, it turns out that these are going to be in a strongly connected component, and this is going to be a sink SCC. So there might be edges coming in but there are no edges coming out. Now, if we set S′ to be false, that means we set its compliment to be true.

So, we satisfied all of these literals. Now, if we satisfied these literals that means later implications are going to be satisfied, and there are no edges coming out so we don't have to follow any implications out.

Now, it turns out that these two approaches are the same. That's why it works. We can take a sink SCC and set it to true. And at the same time, we can take a source SCC and set it to false. These are compliments of each other. If the component S is sink SCC then its complement set is a source SCC.

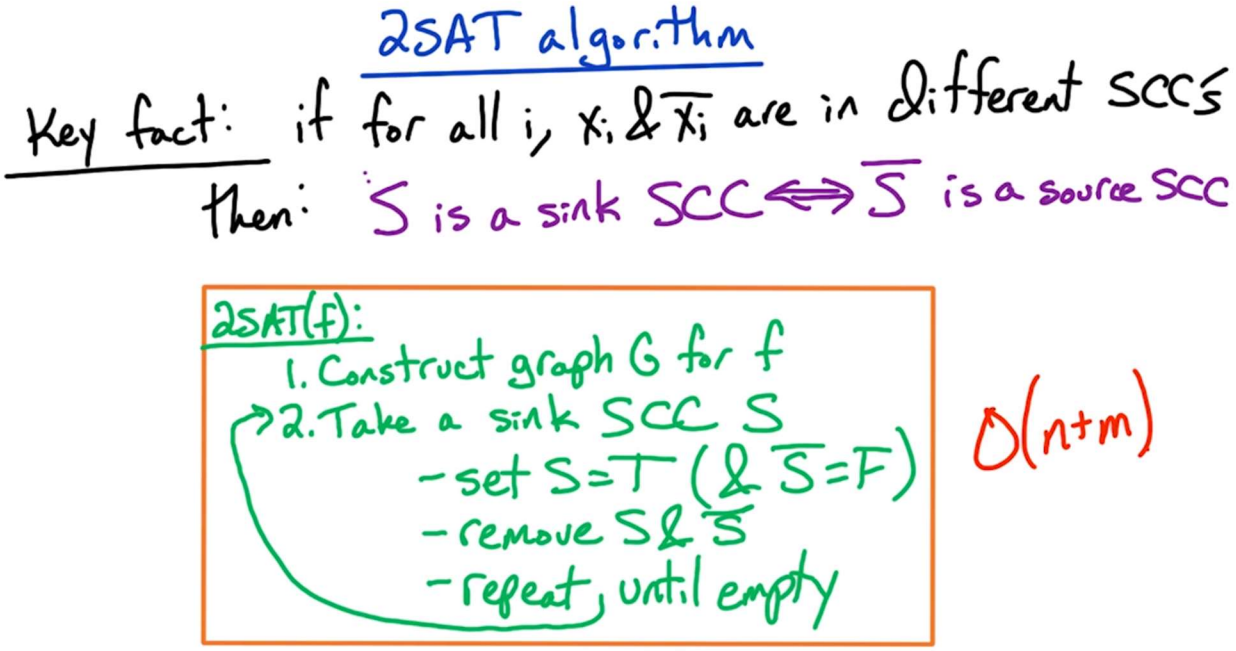
This is what we're going to do, we're going to find a source SCC S′, or we're going to find a sink SCC $\overline{S}$' . These are compliments of each other. If this is a source then this is a sink and vice versa. And we're going to set this source, vertices, to be false.

So these literals are going to be unsatisfied and in the sink we're going to set them all to be true. We're going to satisfy all the literals. If we do one, then we do the other. And once we do this, then we can remove all these literals which appear in S′ and $\overline{S}$'.

Now, for each variable which appears in one of these, this is going to remove the positive and negative literal. If the positive literal appears here, then the negative appears here. And if the negative appears here, then the positive appears here. We remove both the positive and negative literal, so that variable goes away. We can simplify the formula, simplify the graph, and we can repeat.

***

(Slide 11) 2SAT Algorithm




Now the key fact that we just discussed (and we haven't proven but we're going to use it for the algorithm) is that if for all i, xi and $\bar{x}i$ are in different strongly connected components, then we have the important property that if a set of vertices S is a sink SCC, then its compliment is a source SCC and vice versa. If $\bar{S}$ is a source SCC, then S is a sink SCC. So let's take this fact for granted and let's use it to design an algorithm for 2SAT, and then we'll go back and we'll prove this key fact.

Here's our 2SAT algorithm for the input formula f.

1. First, we're going to simplify f, so that we eliminate all unit clauses. So now, we'll assume that f has all clauses of size exactly 2.

   Then, what we do is we construct the graph of implications corresponding to f.

2. Then we run a strongly connected component algorithm on this graph, G. So we have this strongly connected components of G and we have the strongly connected components in topological order.
   - So we're going to take the last component. That's going to be a sink component. Let's call it S. So we're going to satisfy all the literals in this component. If we satisfy f, then that means $\bar{S}$ is unsatisfied.

What are we going to do? We're going to set this component to be true (Set S=True). So we're going to satisfy all the literals in this component. If we satisfy f, then that means $\bar{S}$ is unsatisfied.

So what are we doing? We're taking this sink component and setting it to True. Meanwhile, its complements set is a source component. So we're taking that source component and setting it to false at the same time ($\bar{S} = False$).

- Now, we remove this sink component and this source component

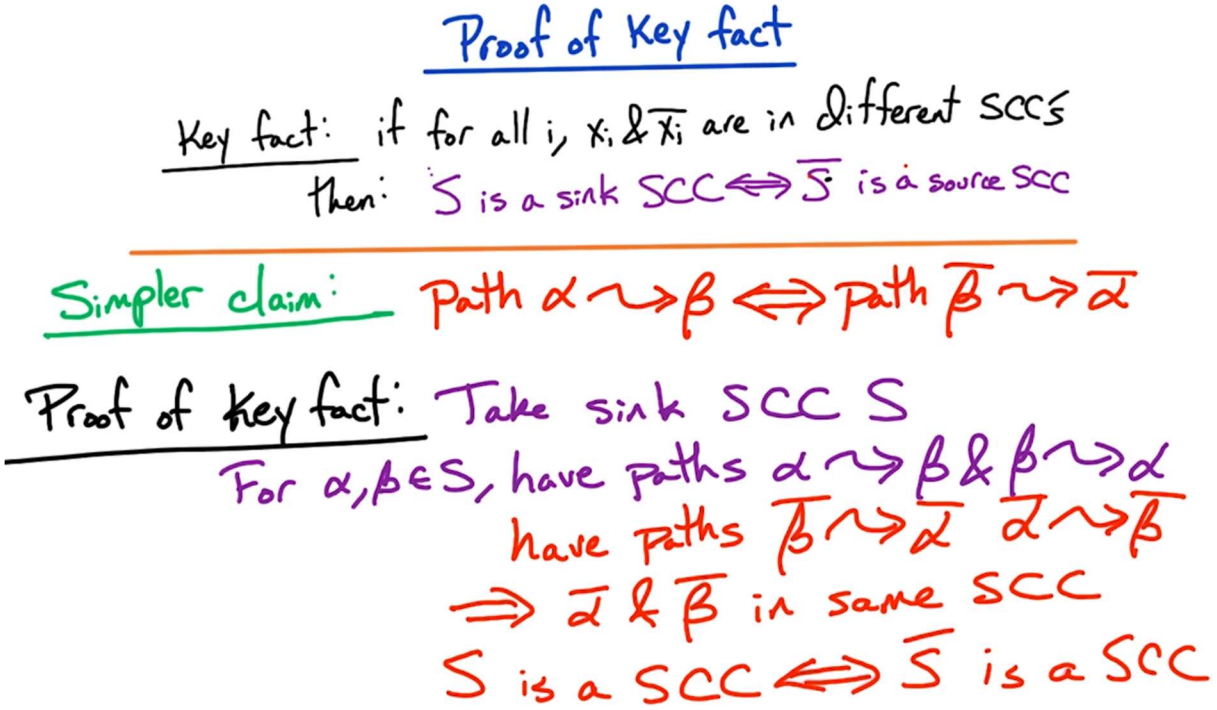
- and we repeat this procedure. We find our sync component in the resulting graph and we keep going until we're left with the empty graph, and then we satisfy the formula.

This completes the algorithm description and it's easy to see that the main step in the algorithm in the running time is constructing the strongly connecting components, which takes linear time. So the whole algorithm takes O(n+m) time. It's linear.

It remains just to prove this key fact, and then we're done with our algorithm description.

***

(Slide 12) Proof of Key Fact




It remains to prove this key fact: that, if S is a sink component then $\bar{S}$, its complement, is a source component and vice versa. If $\bar{S}$ is a source then S is a sink. In order to prove this key fact, we're going to look at a simpler claim.

Here's the claim. If we have a pair of literals $\alpha$ and $\beta$ and if there is a path $\alpha$ -> $\beta$, then there's a path $\bar{\beta}$ -> $\bar{\alpha}$ and if there's a path $\bar{\beta}$ -> $\bar{\alpha}$ , then there's a path $\alpha$ -> $\beta$. Now this claim looks much easier to prove.

Let's assume the claim for now and let's go back and prove this key fact: Let's prove that if S is a sink, then $\bar{S}$ is a source and vice versa.

So, take a sink component S. We're going to prove that $\bar{S}$ is a source component. Now take a pair of vertices in this sink component S. These vertices correspond to literals $\alpha$ and $\beta$. Let's say $\alpha$ and $\beta$ are in S.

So what do we know? We know that $\alpha$ and $\beta$ are strongly connected to each other because they're in the same strongly connected component. If they're strongly connected with each other that means we have paths between the pair. So, we have a path $\alpha$ -> $\beta$ and a path $\beta$ -> $\alpha$. That's the definition of being strongly connected.

Now what does our claim imply? - well if there is a path from $\alpha$ to $\beta$ which is what we know, then there is a path from $\bar{\beta}$ to $\bar{\alpha}$ and this path from $\beta$ to $\alpha$, again applying the claim then that means that we have a path from $\bar{\alpha}$ to $\bar{\beta}$. So what does that mean? That means that $\bar{\alpha}$ from $\bar{\beta}$ are strongly connected with each other – So, they lie in the same SCC.

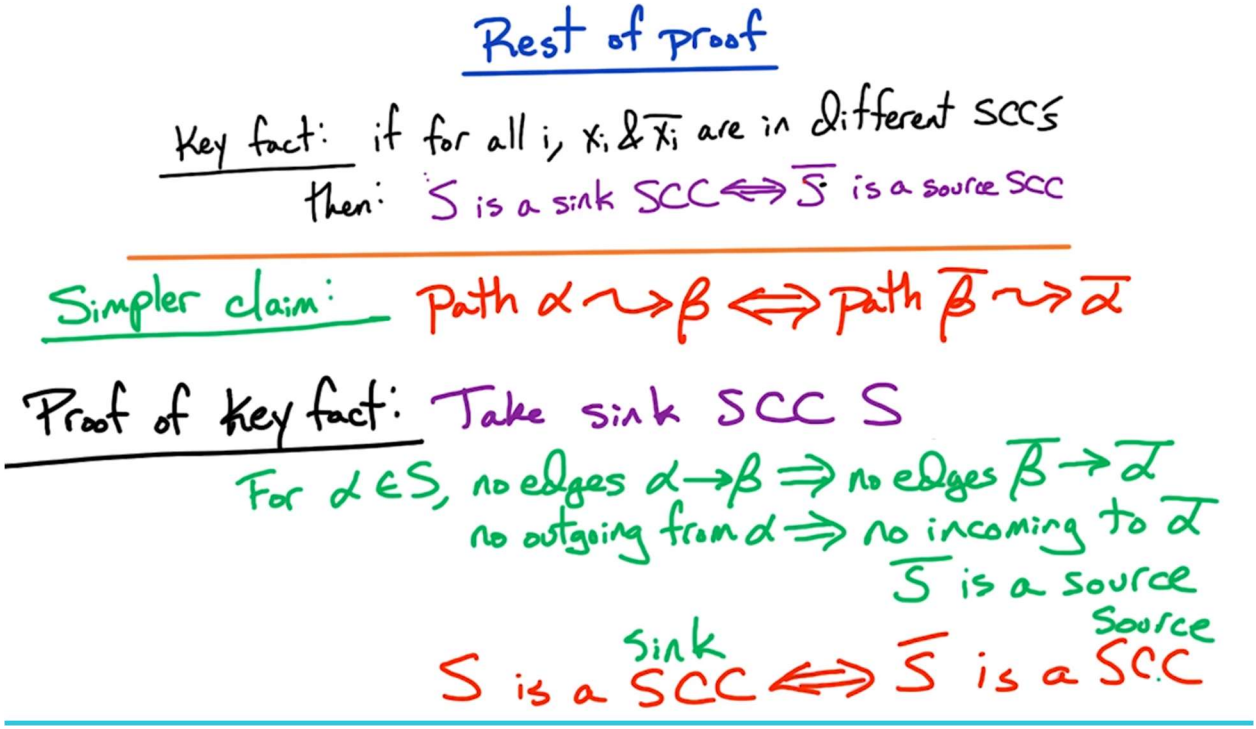
So, if $\alpha$ and $\beta$ are in S then we know that $\bar{\alpha}$ and $\bar{\beta}$ are in $\bar{S}$, an SCC. What we've shown is for pairs of vertices in S, then their complements form an SCC. For $\alpha$ and $\beta$ in S, $\bar{\alpha}$ and $\bar{\beta}$ must be in the same SCC. So, if all pairs over here are strongly connected, then their complements are strongly connected. So, if these guys form an SCC, their complement forms an SCC. Now this is part of what we wanted to prove.

We wanted to prove that if S is a sink SCC then $\bar{S}$ is a source SCC. What have we shown so far? We've shown that if S is itself a SCC then as $\bar{S}$ is an SCC; but, we haven't shown anything about a sink implies a source and vice versa - But we're halfway there! We've shown that if S is an SCC, then its complement is an SCC and vice versa. You can do the same proof in reverse.

And now if $\bar{S}$ is a SCC then its complement is also a SCC. Now let's prove the last part about sink => source and vice-versa.

***

(Slide 13) Rest of Proof




Now let's finish off the rest of the proof of this key fact. What have we shown so far?

We've shown that if we take a sink SCC S, then its complement is a SCC. Now we have to show that $\bar{S}$ is also a source SCC. We have to show that if this is a sink, then this is a source.

Now, S is a sink SCC. If we take a vertex, a literal in S, then we know that $\alpha$ has no outgoing edges - there's no edges of the form $\alpha$ to $\beta$ for any $\beta$. If there are no edges from $\alpha$ to $\beta$, then what does the claim imply? - that implies that there are no edges into $\bar{\alpha}$.

So, if there are no outgoing edges from alpha, then there are no incoming edges to $\bar{\alpha}$. So, if, for all alpha in S, there are no incoming edges to $\bar{\alpha}$, then that means that $\bar{S}$ has no incoming edges. So that means that as $\bar{S}$ is a source SCC.

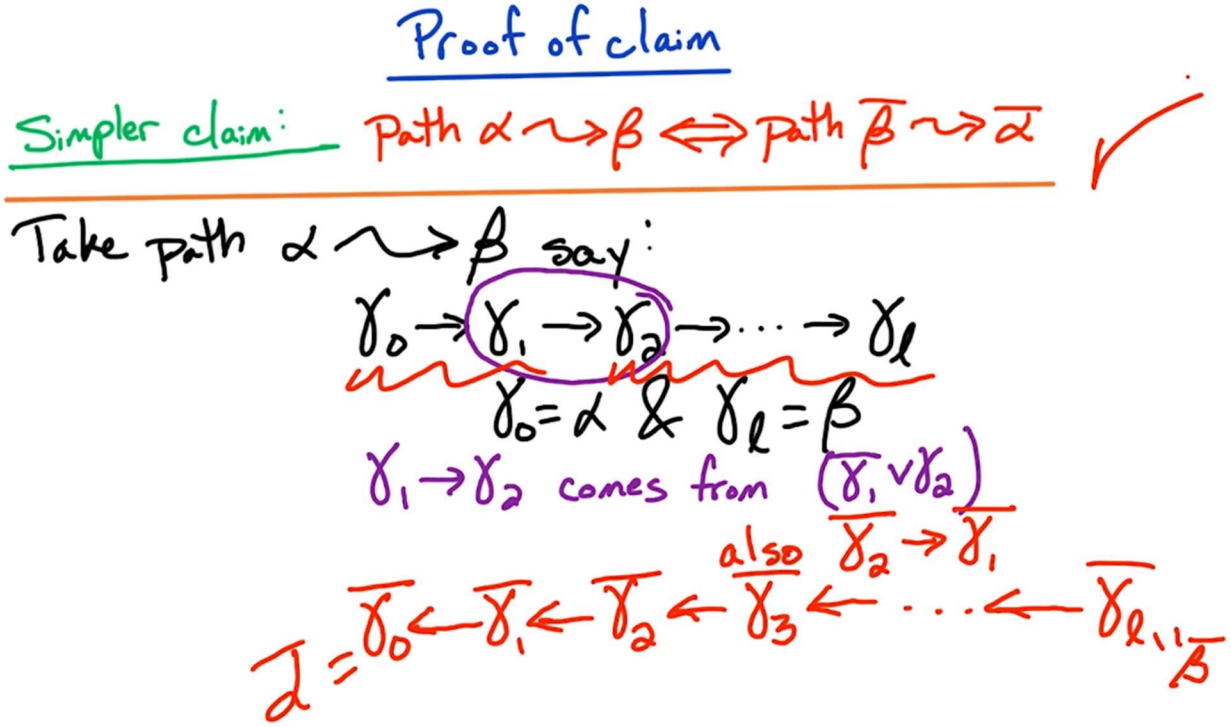
So we've shown that if S is a sink SCC, then by this claim, $\bar{S}$ must be a source SCC.

And we can do the argument in reverse and show that if $\bar{S}$ is a source, then S is a sink SCC. So we've shown that if S is a sink SCC, then $\bar{S}$ S is a source SCC. And that proves the key fact.

Now it just remains to prove this claim.

***

(Slide 14) Proof of Claim (Satisfiability)




Now let's prove this claim that, if there is a path from $\alpha$ to $\beta$, there's a path from $\bar{\beta}$ to $\bar{\alpha}$, and vice versa. So let's take a path from $\alpha$ to $\beta$ and let's construct a path from $\bar{\beta}$ to $\bar{\alpha}$.

Let's give some notation for the vertices along this path from $\alpha$ to $\beta$. Let's say this path from $\alpha$ to $\beta$ goes along $\gamma 0$ to $\gamma 1$ and so on up to $\gamma l$. So it starts at $\alpha$, so $\gamma 0 = \alpha$ and $\gamma l = \beta$.

Now let's take one of these edges. Let's look at the edge from $\gamma 1$ to $\gamma 2$. How do we get an edge from $\gamma 1$ to $\gamma 2$? - well, this edge from $\gamma 1$ to $\gamma 2$ comes from this clause, $\overline{\gamma 1} \vee \gamma 2$.

Now every clause has two edges which it implies. This is one of the edges. What's the other edge? The other edge we get, the other implication is $\overline{\gamma 2}$ -> $\overline{\gamma 1}$. Similarly, if we look at this edge $\gamma 0$ -> $\gamma 1$, we're going to see that there's also an edge from $\overline{\gamma 1}$ -> $\overline{\gamma 0}$.

And if we look along this part of the path, we're going to see that we get a path from $\overline{\gamma 1}$ all the way over to $\overline{\gamma 0}$.

What is $\overline{\gamma 1}$ ? This is $\bar{\beta}$. And what is $\overline{\gamma 0}$ ? It's $\bar{\alpha}$. So, what we've shown is that if there is a path from $\alpha$ to $\beta$, then there must be a path from $\bar{\beta}$ to $\bar{\alpha}$, and that's what we wanted to prove.

We wanted to prove that if there is a path from $\alpha$ to $\beta$, then we can show that there is a path from $\bar{\beta}$ to $\bar{\alpha}$. And then we can apply the same argument in reverse, and we can show that if there is a path from $\bar{\beta}$ to $\bar{\alpha}$, and there's a path from $\alpha$ to $\beta$.

That proves the claim, and that completes the proof of correctness of our algorithm. So we've completed the argument for the linear time algorithm for 2-SAT.