CSC363 Tutorial 9

What am I even doing anymore ;-;

Paul “sushi_enjoyer” Zhang

University of Chux

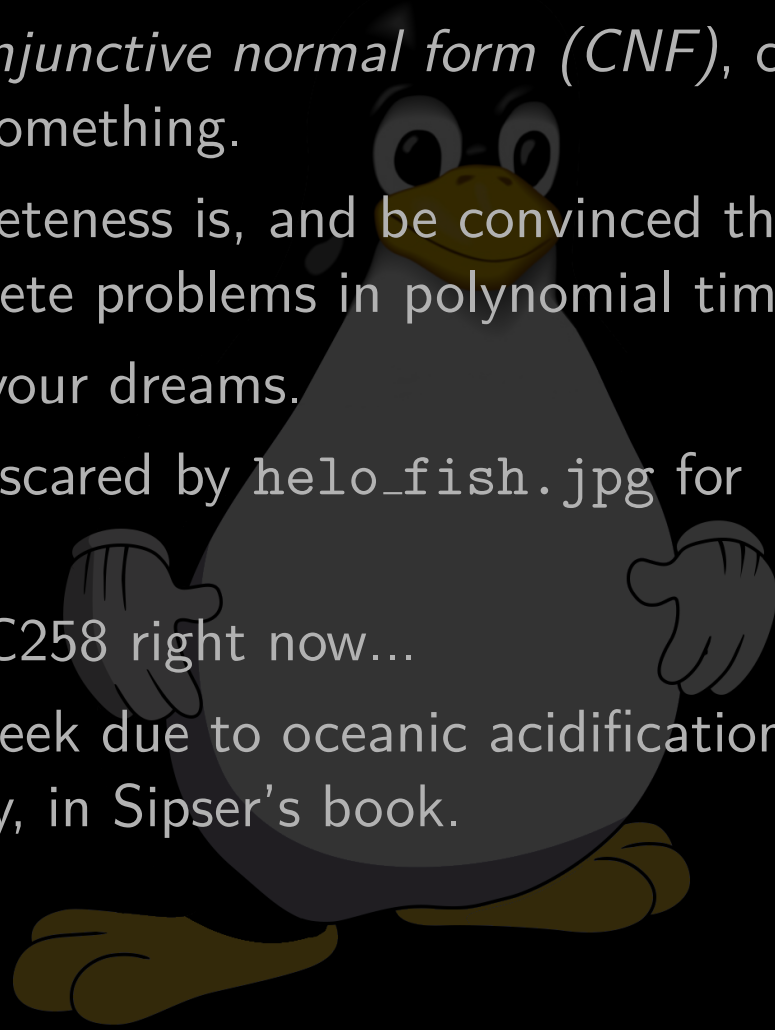
March 17, 2021

Learning objectives this tutorial

By the end of this tutorial, you should...

- Be able to convert *formulas* into *conjunctive normal form (CNF)*, cuz we need it to understand *3SAT* or something.
- Have a brief idea of what NP-completeness is, and be convinced that you shouldn't try to solve NP-complete problems in polynomial time.
- Have the truth tables haunt you in your dreams.
- Rejoice! because, uh, you'll only be scared by `helo_fish.jpg` for only two more weeks?
- Feel uneasy, if you've are taking CSC258 right now...

(`helo_fish.jpg` is taking a break this week due to oceanic acidification.)
Big Chux certified readings: 7.4, probably, in Sipser's book.



AAAAAAAAAAAAAAA

Tutorial

MM

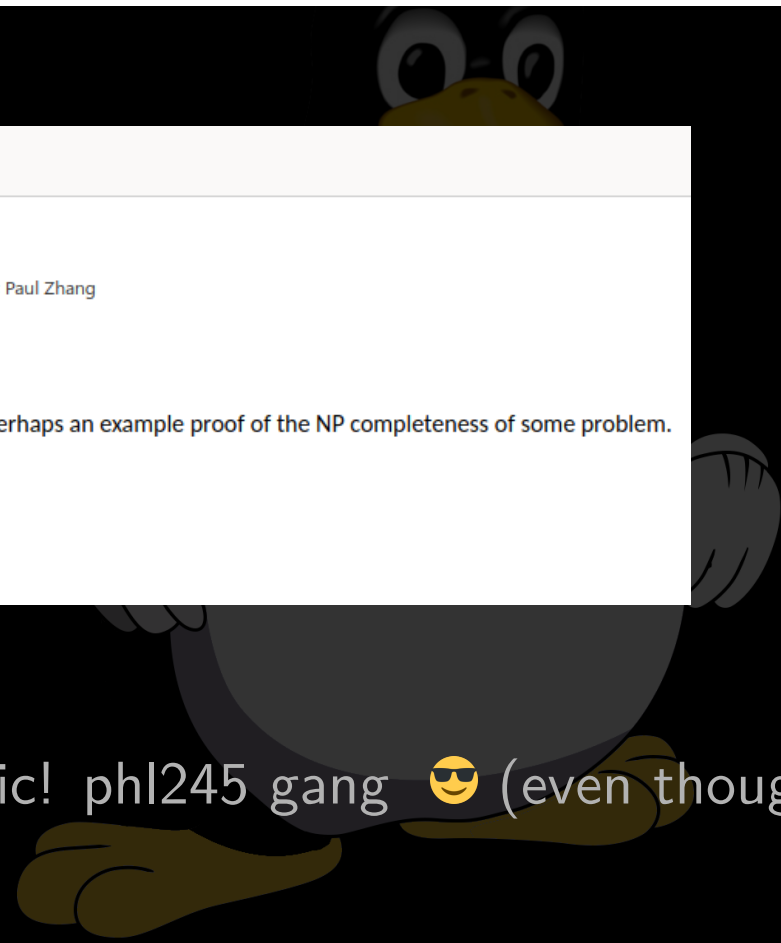
Mohammad Mahmoud
Mon 15/03/2021 21:43
To: Eric Lauw; Yousef Akiba <yousefakiba@gmail.com>; Muhammad Huzaifa; Daniel Ceniceros; Paul Zhang

Dear TAs,

Feel free to present whatever you think is helpful in your tutorial this week. Perhaps an example proof of the NP completeness of some problem.

Thank you

Mohammad



D:

so uh, i hope youse like formulas and logic! phl245 gang 😎 (even though i havent even taken it before)

formula

(You'll see the motivation for this section later! it relates to the *3SAT* problem)

Task: Write out your favourite formula, preferably in LaTeX, in the chat. My favourite formula is

$$|p_D(n) - r_D(n)| \leq \frac{1}{n^c}.$$

formula

uh, today we're gonna talk about a different type of formula, probably ;-; we're talking about boolean formulas![1]

Definition: a **boolean formula** (or **formula**) is any valid expression involving a bunch of "boolean variables" and the operations $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$, and brackets. [2]

Task: Which of the following are (in your opinion) boolean formulas?

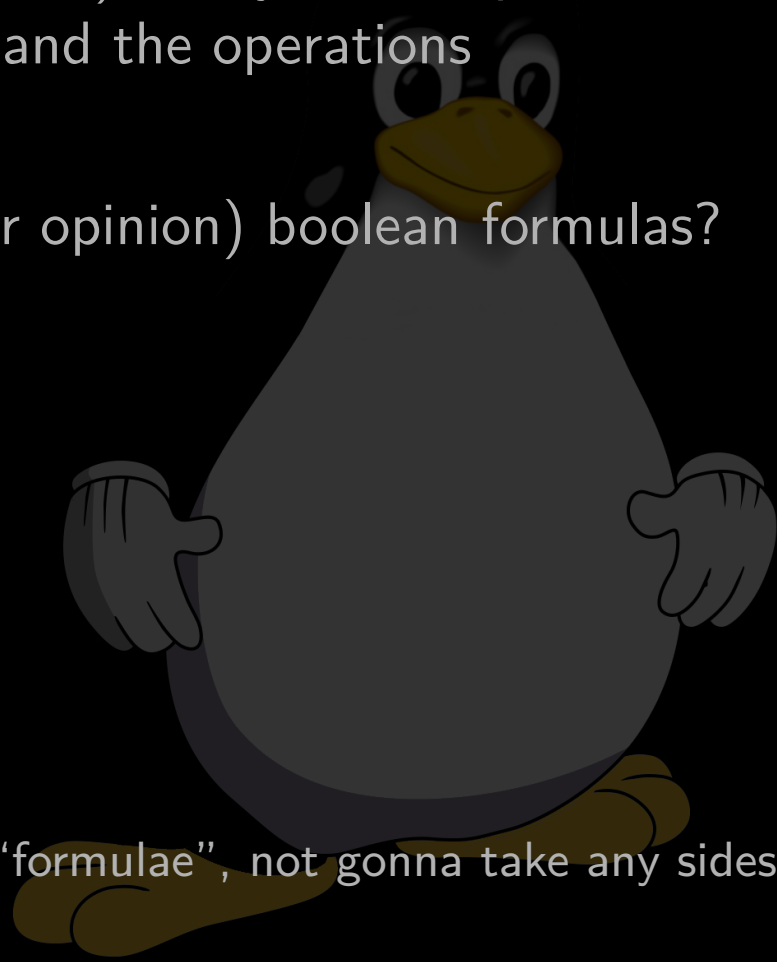
(a) $x_1 = x_2$.

(b) $(x \Rightarrow y) \Leftrightarrow \neg(y \wedge z)$.

(c) $((x \wedge y) \Rightarrow (x)$.

(d) $\Rightarrow$ 🍣 $\Rightarrow$ 🧃 $\Rightarrow$ 🗿.

Answer: only (b) is a formula.



[1] idk if the plural of formula is "formulas" or "formulae", not gonna take any sides here.

[2] It's kinda an informal definition, but we'll just work with it for now. please accept it.

formula

uh, today we're gonna talk about a different type of formula, probably ;-; we're talking about boolean formulas![1]

Definition: a **boolean formula** (or **formula**) is any valid expression involving a bunch of "boolean variables" and the operations $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$, and brackets. [2]

Task: Which of the following are (in your opinion) boolean formulas?

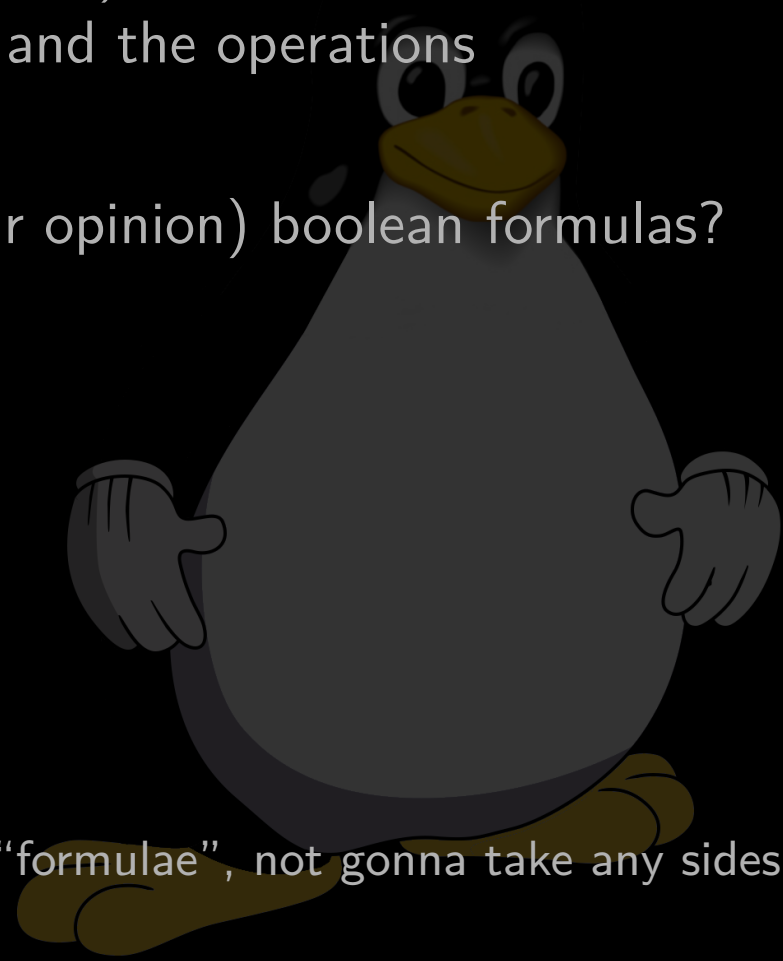
(a) $x_1 = x_2$.

(b) $(x \Rightarrow y) \Leftrightarrow \neg(y \wedge z)$.

(c) $((x \wedge y) \Rightarrow (x)$.

(d) $\Rightarrow$  $\Rightarrow$  $\Rightarrow$  .

Answer: only (b) is a formula.



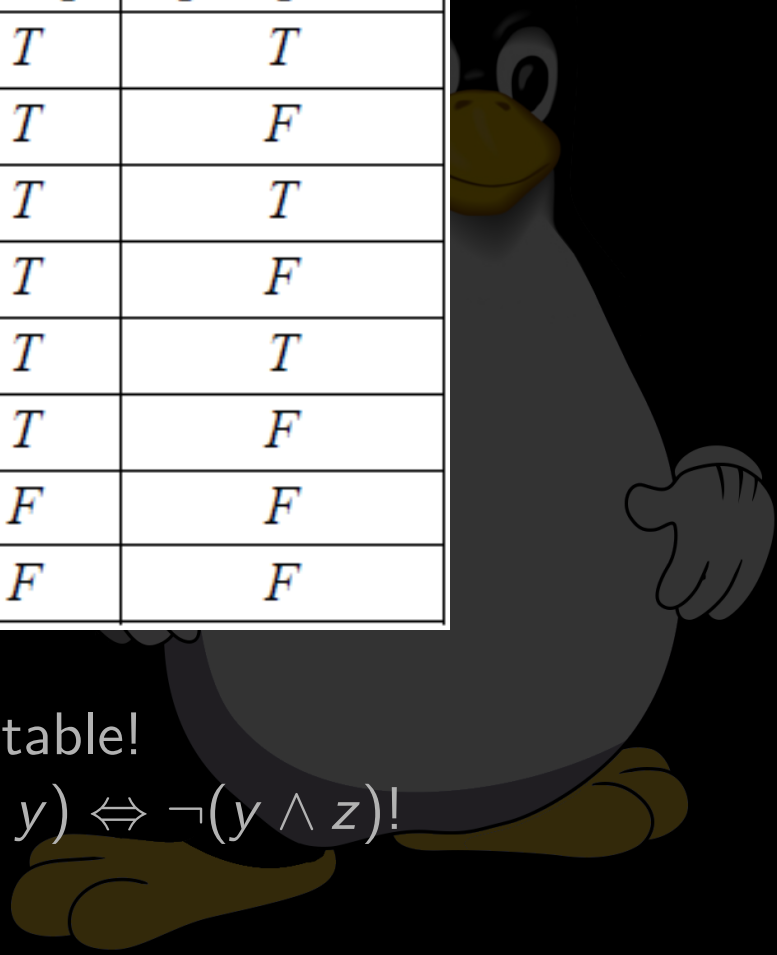
[1] idk if the plural of formula is "formulas" or "formulae", not gonna take any sides here.

[2] It's kinda an informal definition, but we'll just work with it for now. please accept it.

formula

Do you remember truth tables?

p	q	r	$p \vee q$	$(p \vee q) \wedge r$
T	T	T	T	T
T	T	F	T	F
T	F	T	T	T
T	F	F	T	F
F	T	T	T	T
F	T	F	T	F
F	F	T	F	F
F	F	F	F	F

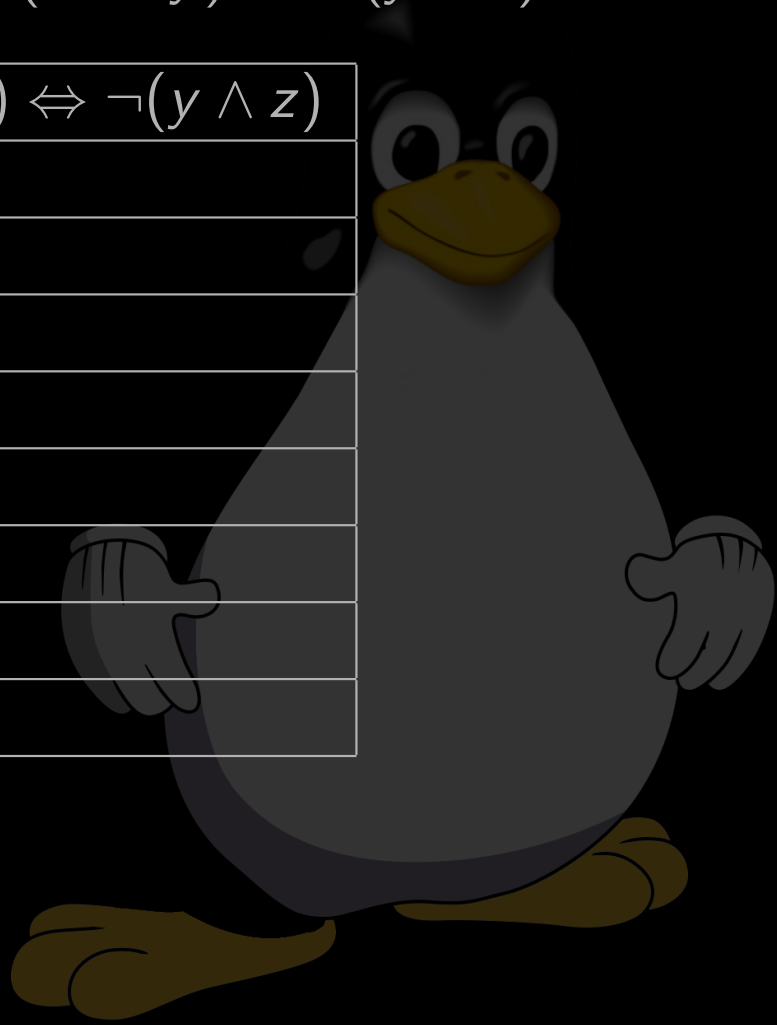


Given a formula, we can fill out its truth table!
Let us write out the truth table for $(x \Rightarrow y) \Leftrightarrow \neg(y \wedge z)$!

formula

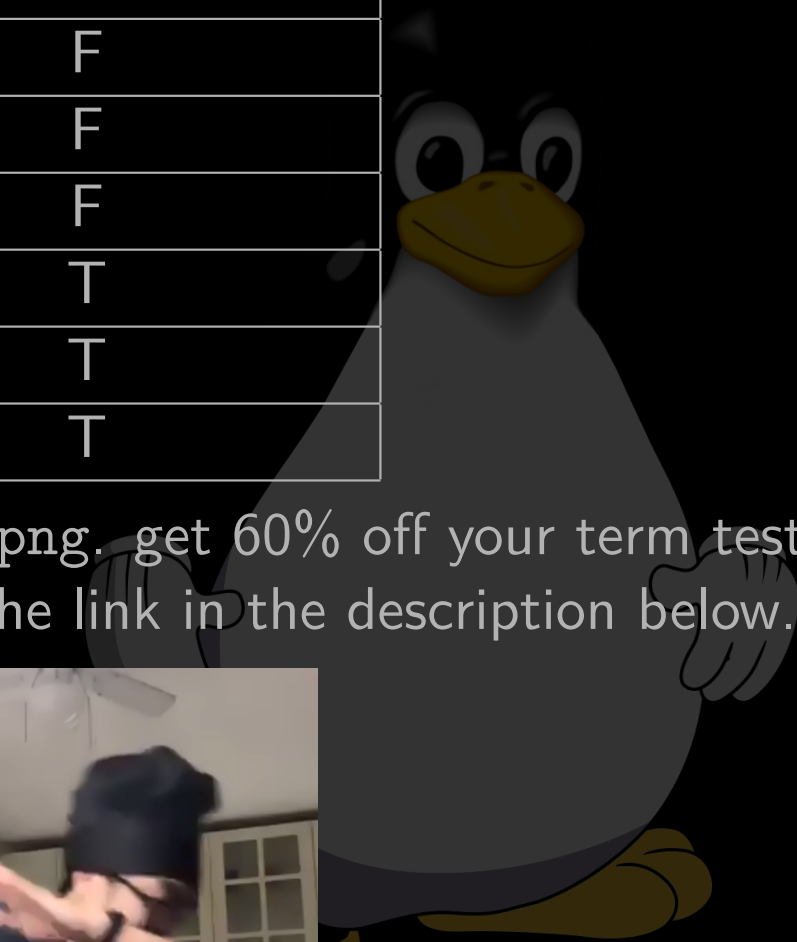
Task: Fill in the following truth table for $(x \Rightarrow y) \Leftrightarrow \neg(y \wedge z)$.

x	y	z	$(x \Rightarrow y) \Leftrightarrow \neg(y \wedge z)$
T	T	T	
T	T	F	
T	F	T	
T	F	F	
F	T	T	
F	T	F	
F	F	T	
F	F	F	



formula

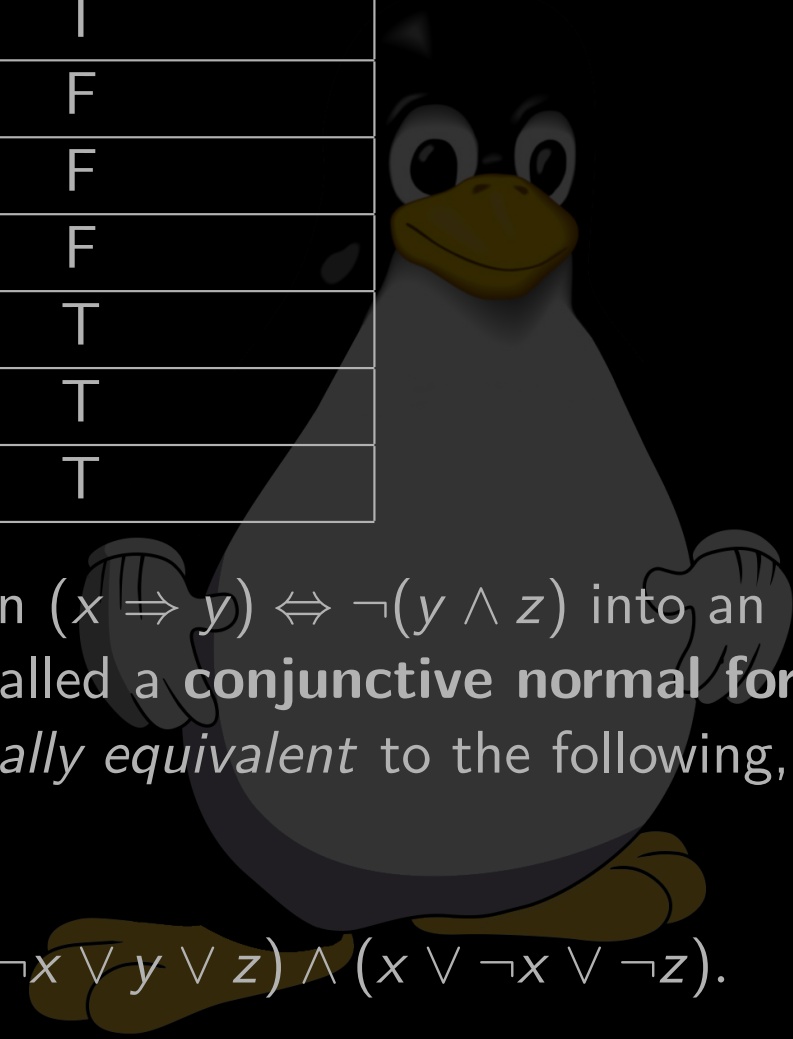
x	y	z	$(x \Rightarrow y) \Leftrightarrow \neg(y \wedge z)$
T	T	T	F
T	T	F	T
T	F	T	F
T	F	F	F
F	T	T	F
F	T	F	T
F	F	T	T
F	F	F	T



Today's tutorial is sponsored by `larry.png`. get 60% off your term test grade with code `MAXTERM` today. Click the link in the description below.

formula

x	y	z	$(x \Rightarrow y) \Leftrightarrow \neg(y \wedge z)$
T	T	T	F
T	T	F	T
T	F	T	F
T	F	F	F
F	T	T	F
F	T	F	T
F	F	T	T
F	F	F	T



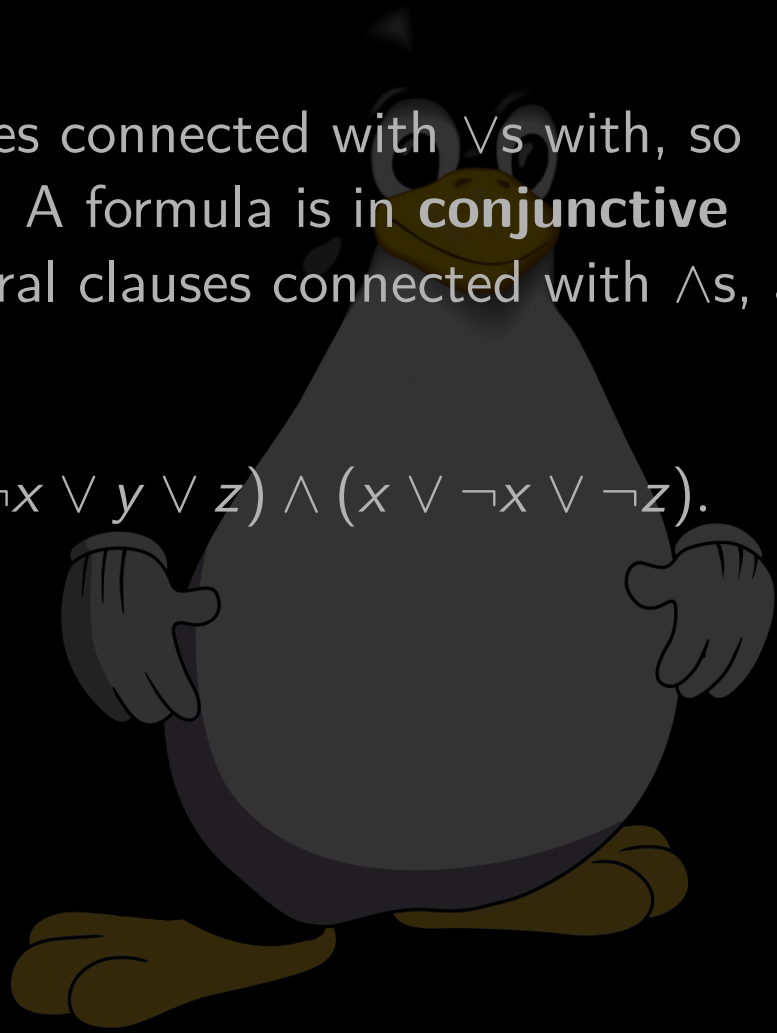
Anyway, this truth table allows us to turn $(x \Rightarrow y) \Leftrightarrow \neg(y \wedge z)$ into an equivalent statement written in what's called a **conjunctive normal form (CNF)**. So $(x \Rightarrow y) \Leftrightarrow \neg(y \wedge z)$ is *logically equivalent* to the following, which is in CNF:

$$(\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg x \vee \neg z).$$

formula

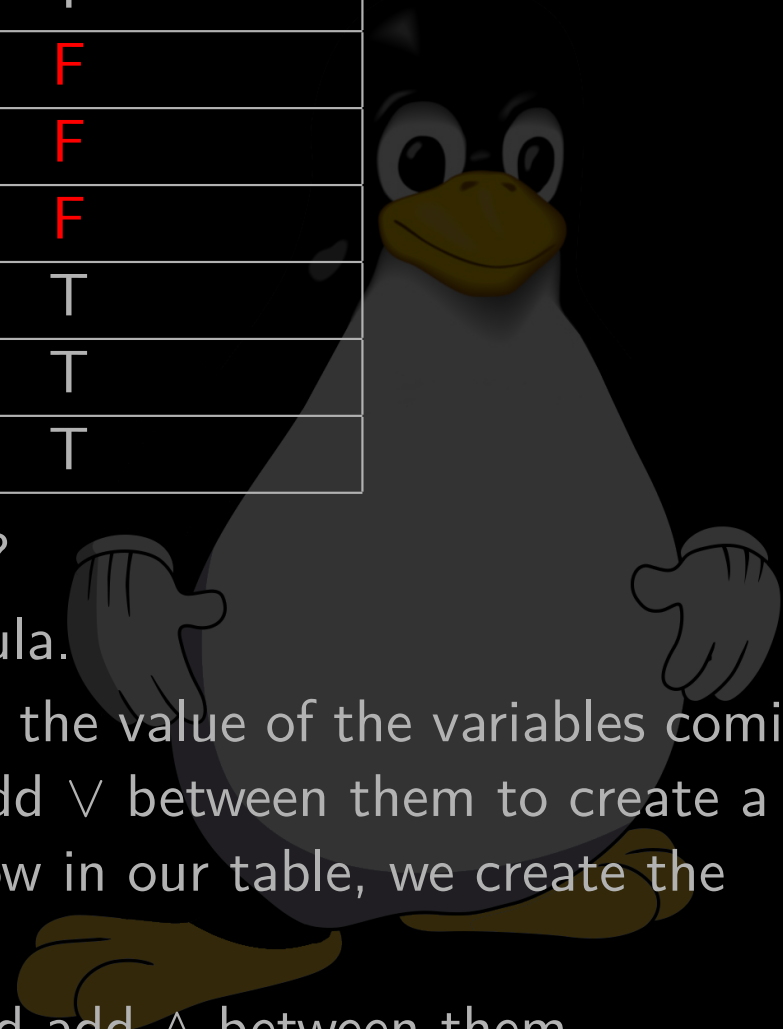
More formally, a **clause** is several variables connected with $\vee$s with, so something like $(x \vee \neg y \vee \neg z)$ is a clause. A formula is in **conjunctive normal form (CNF)** if it comprises several clauses connected with $\wedge$s, as in

$$(\neg x \vee \neg y \vee \neg z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg x \vee \neg z).$$



formula

x	y	z	$(x \Rightarrow y) \Leftrightarrow \neg(y \wedge z)$
T	T	T	F
T	T	F	T
T	F	T	F
T	F	F	F
F	T	T	F
F	T	F	T
F	F	T	T
F	F	F	T



How to write out the CNF for a formula?

(1) Fill out the truth table for the formula.

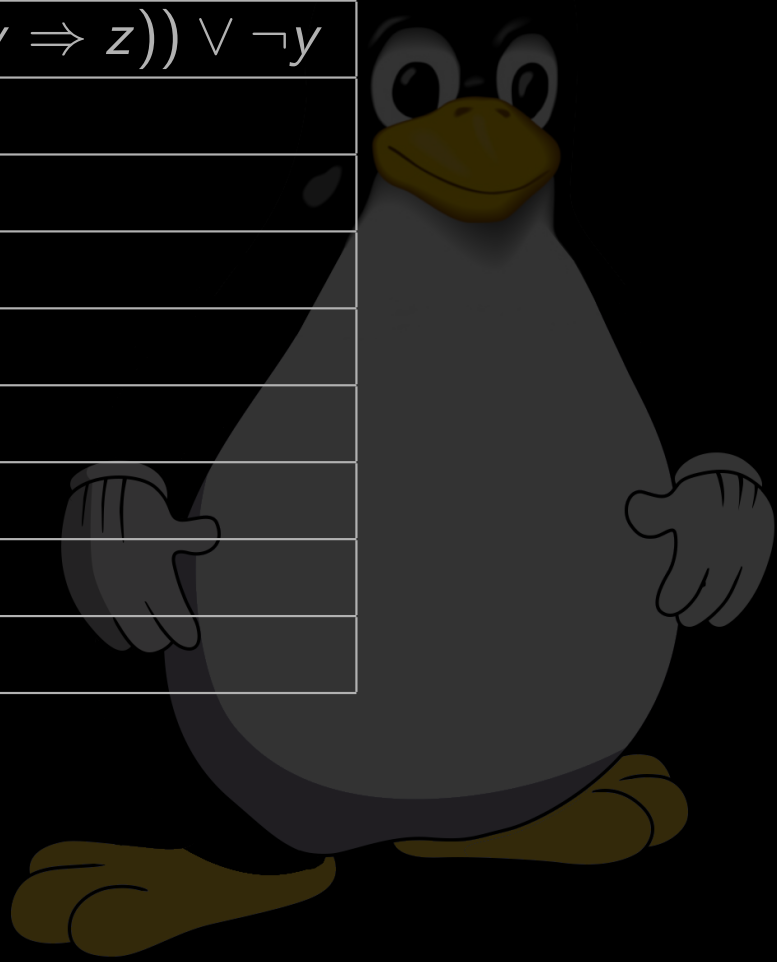
(2) For each row which ends in 'F', take the value of the variables coming before it, and negate them. Then add $\vee$ between them to create a clause. For example, for the third row in our table, we create the expression $(\neg x \vee y \vee \neg z)$.

(3) Take all the clauses you've made and add $\wedge$ between them.

formula

Task: Write out the CNF for the formula $(x \Leftrightarrow (y \Rightarrow z)) \vee \neg y$.

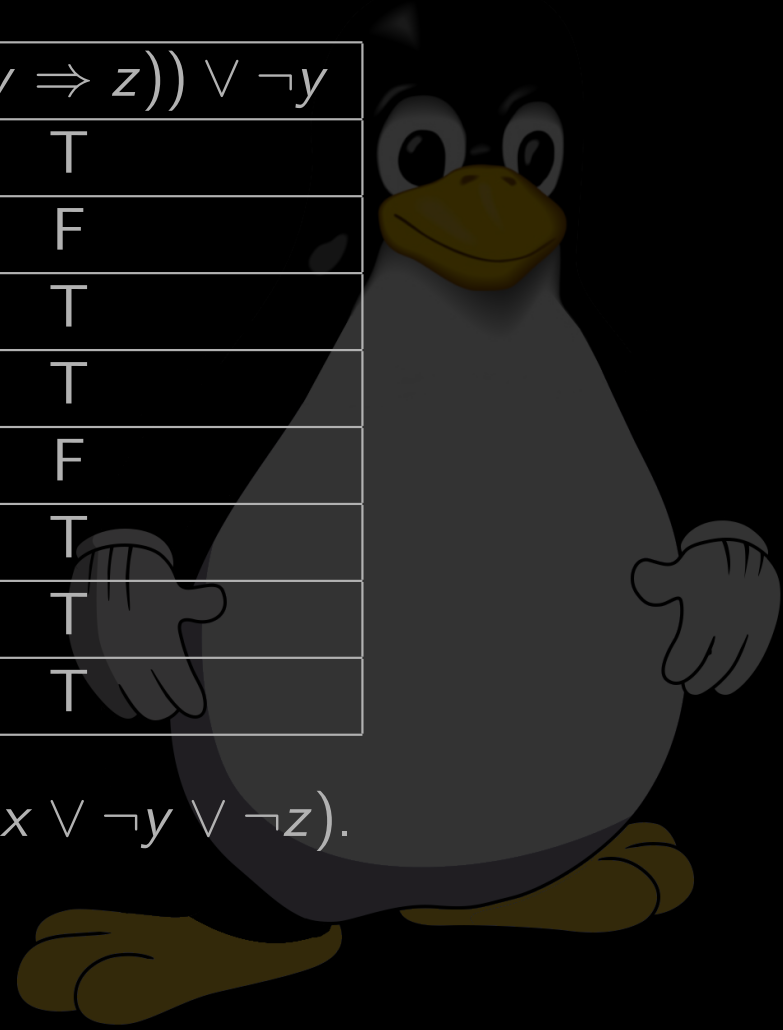
x	y	z	$(x \Leftrightarrow (y \Rightarrow z)) \vee \neg y$
T	T	T	
T	T	F	
T	F	T	
T	F	F	
F	T	T	
F	T	F	
F	F	T	
F	F	F	



formula

Task: Write out the CNF for the formula $(x \Leftrightarrow (y \Rightarrow z)) \vee \neg y$.

x	y	z	$(x \Leftrightarrow (y \Rightarrow z)) \vee \neg y$
T	T	T	T
T	T	F	F
T	F	T	T
T	F	F	T
F	T	T	F
F	T	F	T
F	F	T	T
F	F	F	T



So the CNF should be $(\neg x \vee \neg y \vee z) \wedge (x \vee \neg y \vee \neg z)$.

are you satisfied 😳

Now we introduce the **3SAT problem**. Say you are given a formula in CNF, but in each bracketed term there must be exactly 3 variables. So

$(x \Leftrightarrow (y \Rightarrow z)) \vee \neg y$ doesn't work, cuz it's not in CNF.

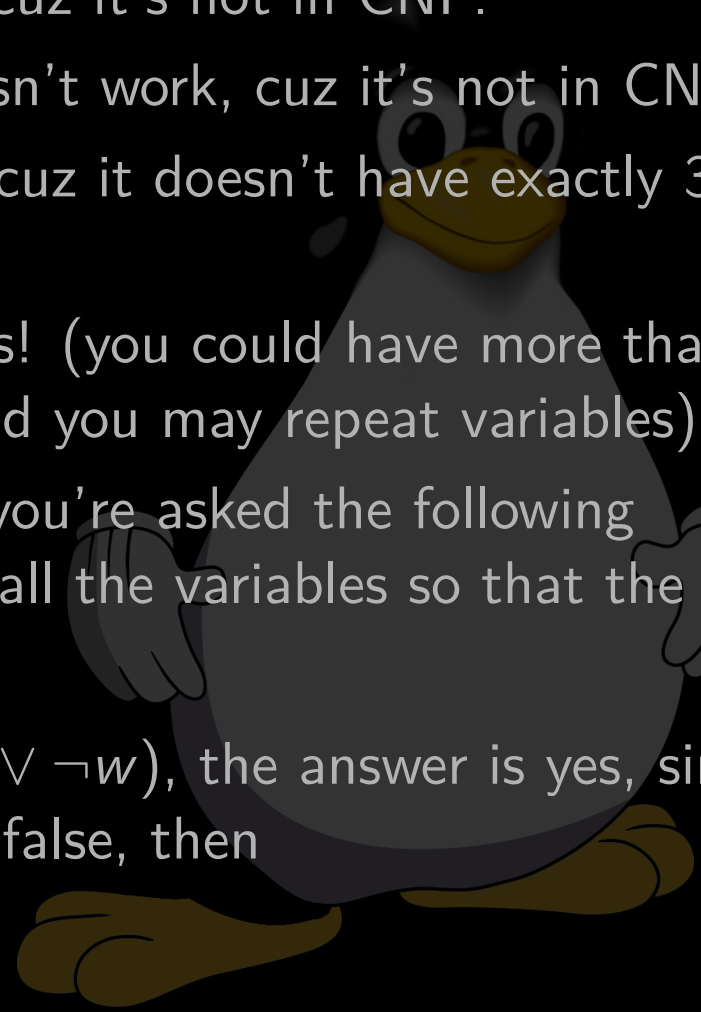
$(x \wedge y \wedge \neg z) \vee (\neg x \wedge \neg y \wedge \neg z)$ doesn't work, cuz it's not in CNF.

$(x \vee y \vee z) \wedge (y \vee z)$ doesn't work, cuz it doesn't have exactly 3 variables in each bracket.

$(x \vee \neg y \vee x) \wedge (\neg x \vee z \vee \neg w)$ works! (you could have more than 3 variables in the whole expression, and you may repeat variables).

So you're given this CNF formula,[3] and you're asked the following question: can you assign truth values to all the variables so that the CNF formula is true?

In the example $(x \vee \neg y \vee \neg z) \wedge (\neg x \vee z \vee \neg w)$, the answer is yes, since if we assign x, z to be true and y, w to be false, then $(x \vee \neg y \vee \neg z) \wedge (x \vee z \vee \neg w)$ is true.



[3] it's in fact called a 3CNF formula!

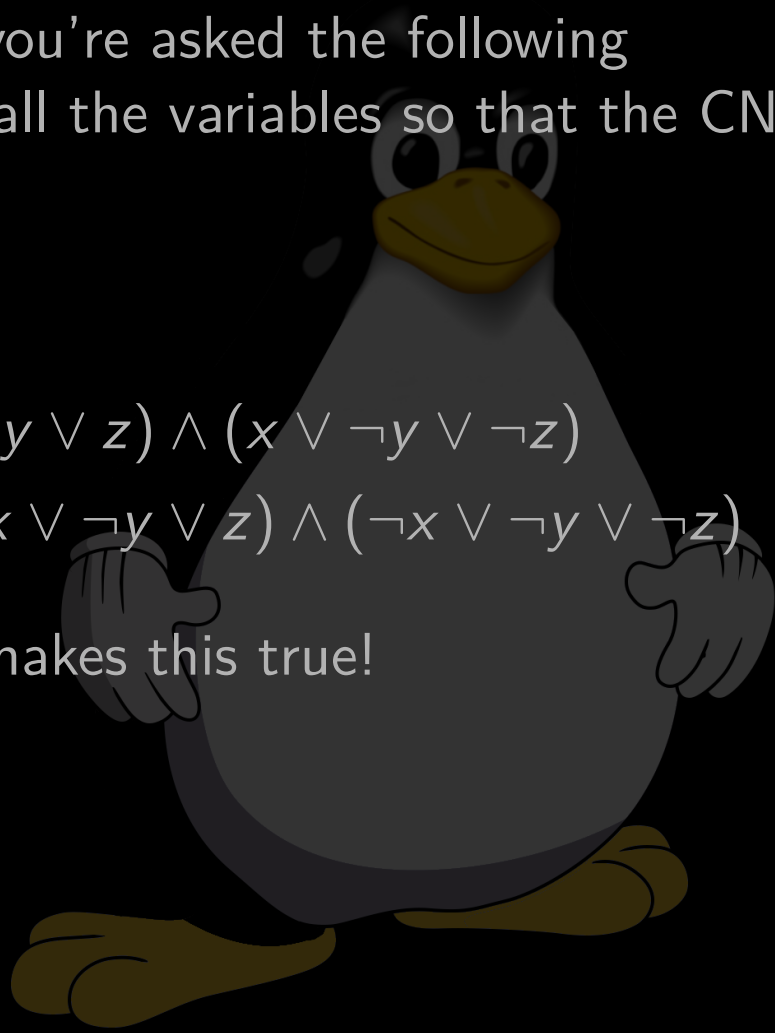
are you satisfied 😳

So you're given this CNF formula,[4] and you're asked the following question: can you assign truth values to all the variables so that the CNF formula is true?

In the example

$$(x \vee y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee z) \wedge (x \vee \neg y \vee \neg z)$$
$$\wedge (\neg x \vee y \vee z) \wedge (\neg x \vee y \vee \neg z) \wedge (\neg x \vee \neg y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$$

there is no assignment of variables that makes this true!



[4] it's in fact called a 3CNF formula!

are you satisfied 😳

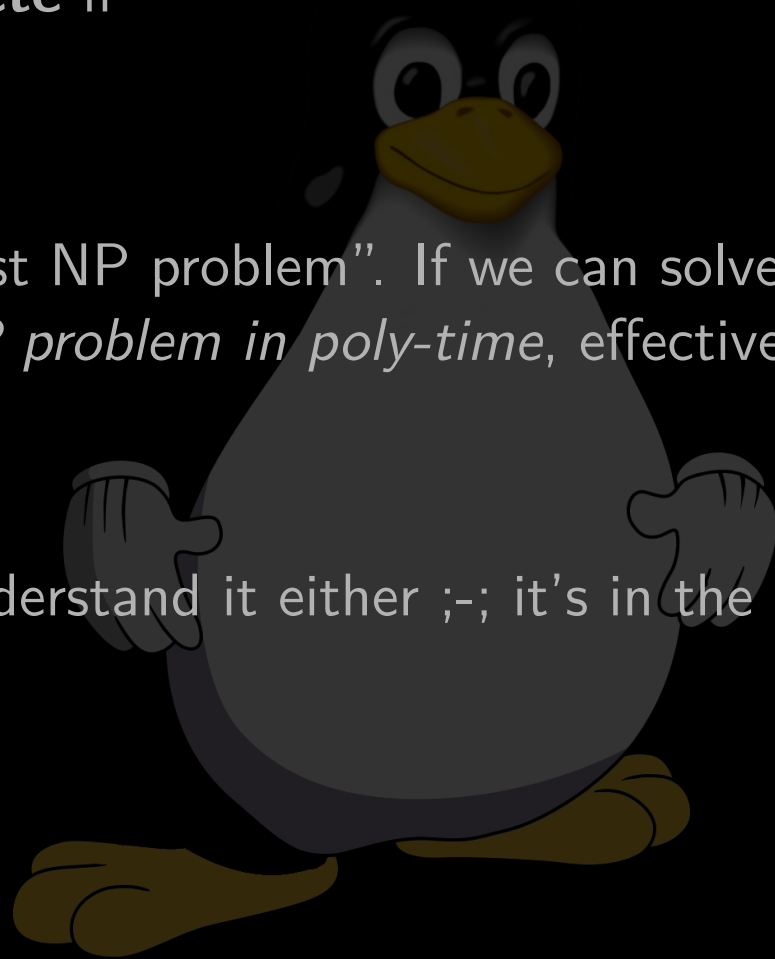
Turns out, uh, 3SAT[5] is what is called a *NP-complete problem*.
Definition: A language A is **NP-complete** if

(a) $A \in \text{NP}$;

(b) For every $B \in \text{NP}$, $B \leq_p A$.

The above is saying that A is the "hardest NP problem". If we can solve A in poly-time, then we can solve *every NP problem in poly-time*, effectively proving $\text{P} = \text{NP}$.

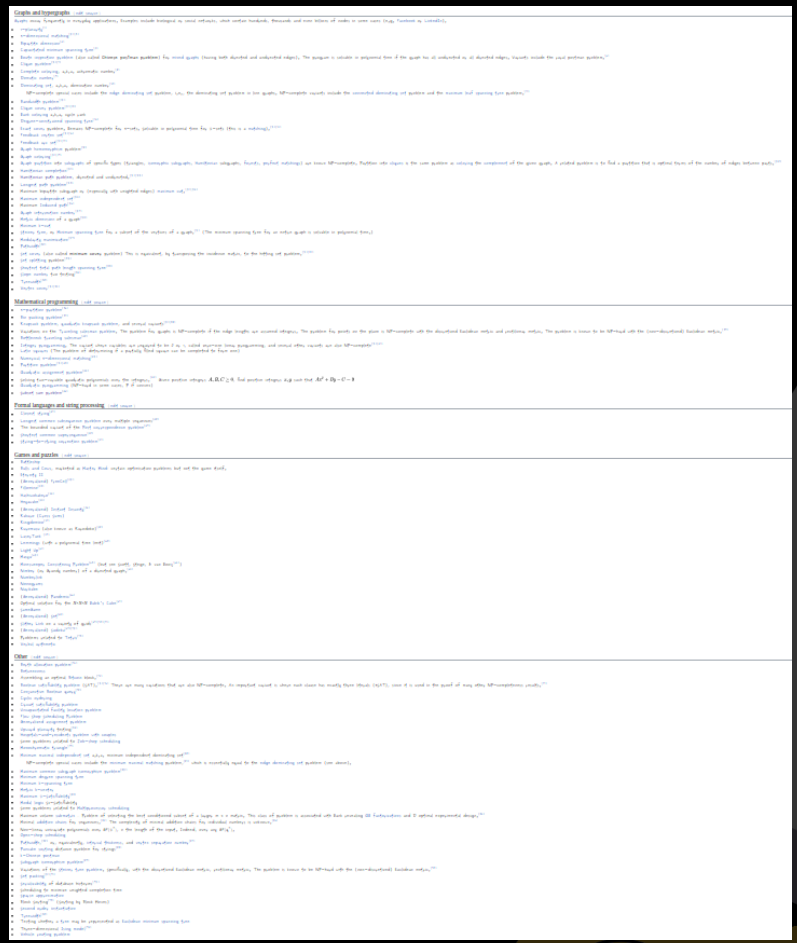
Theorem: 3SAT is NP-complete.
Proof: sorry it's too long and i don't understand it either ;-; it's in the book tho!



[5] As a language, 3SAT is defined as $\{\varphi : \varphi \text{ is a 3CNF formula that can be satisfied}\}$.

np-complete

3SAT is not the *only* NP-complete problem though! There's a whole plethora of them. They are all *the hardest problems in NP*. If you can solve any one of them in poly-time, then all of them can be solved in poly-time.

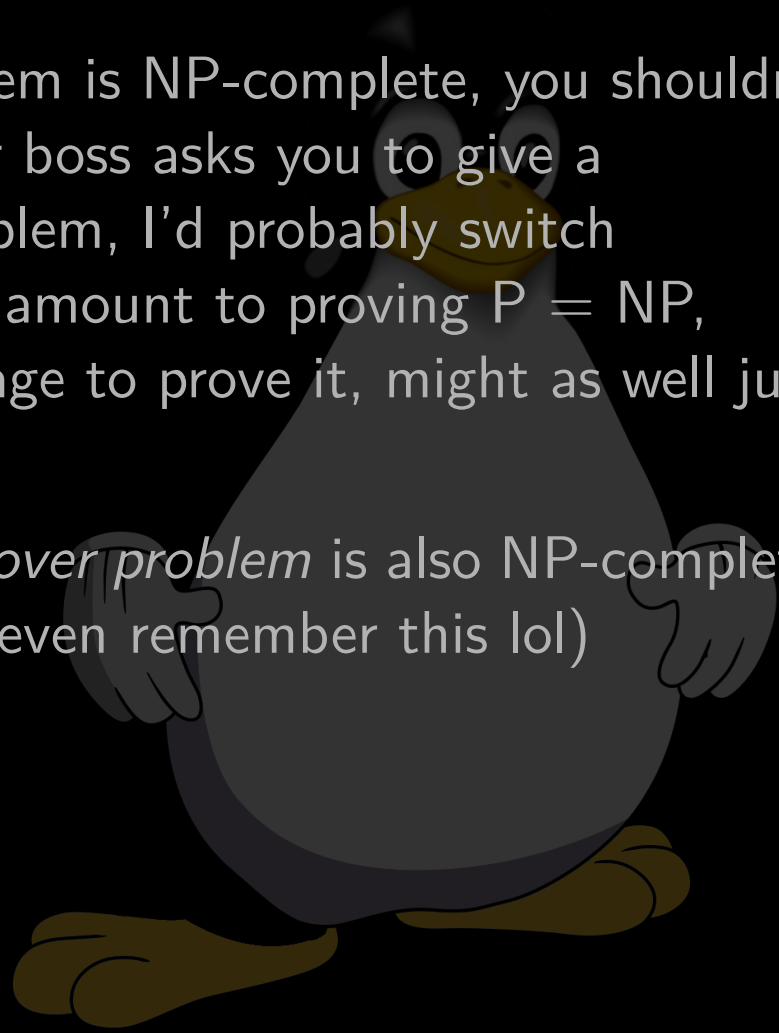


List of NP-complete problems

np-complete

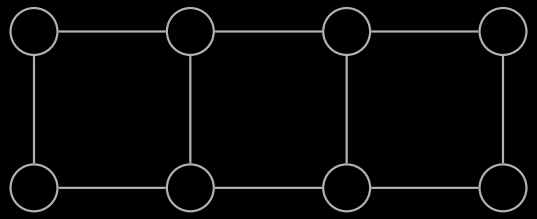
Basically, if you manage to prove a problem is NP-complete, you shouldn't try to solve it in polynoimal time! If your boss asks you to give a poly-time solution to a NP-complete problem, I'd probably switch employers... cuz that would pretty much amount to proving P = NP, which is really hard! (and if you do manage to prove it, might as well just walk away with the million dollars)

We'll now prove explain that the *vertex cover problem* is also NP-complete. If you've been in csc373, rejoice! (if you even remember this lol)

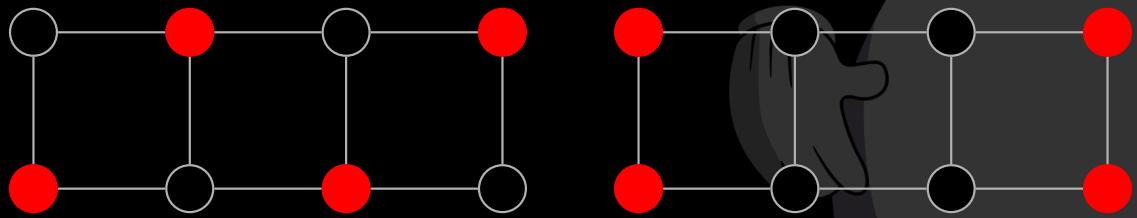


cover me owo 😳

Let's say you're given an undirected graph G. something like



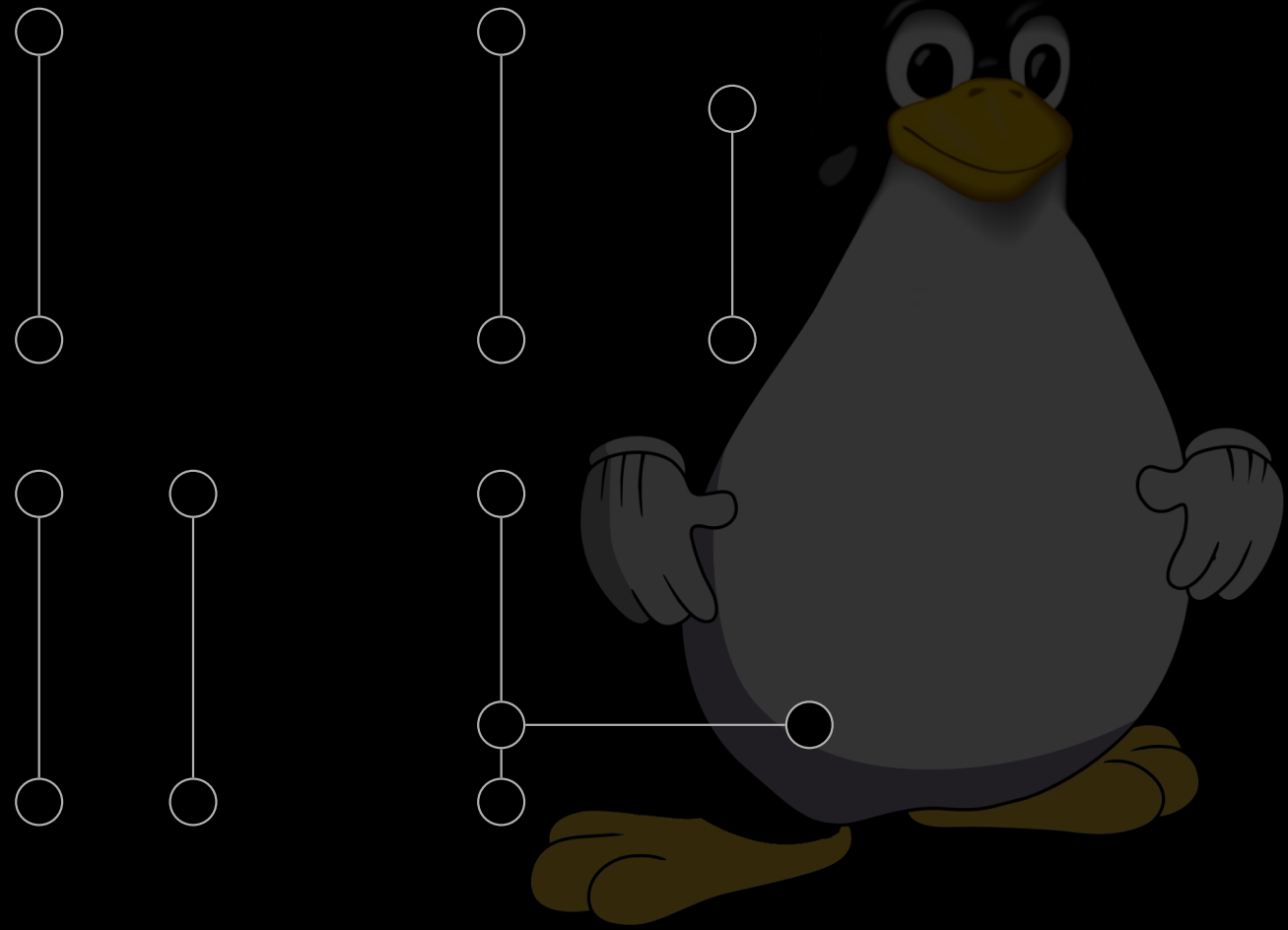
A **vertex-cover** for G is a set of vertices V in G such that every edge in G touches at least one vertex in V. In the graph on the left below, the red vertices form a vertex cover; in the graph on the right, the red vertices don't form a vertex cover.



The **size** of a vertex cover is the number of vertices in our vertex cover. The size of the vertex cover on the left graph is 4. (In fact this is the minimum vertex cover size for our graph!)

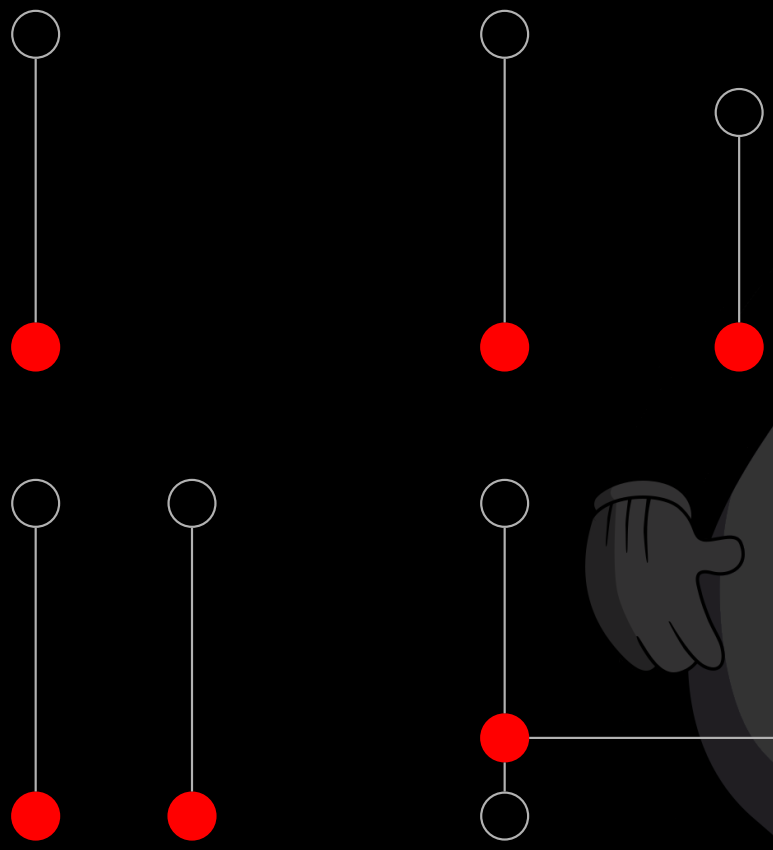
cover me owo 😳

Task: Find a vertex cover for the following graph G (it has four disconnected components). What is the size of the covering? Can you make it smaller?



cover me owo 😳

Task: Find a vertex cover for the following graph G (it has four disconnected components). What is the size of the covering? Can you make it smaller?



(size 6)

cover me owo 😳

The **vertex cover problem** gives you an arbitrary undirected graph G and an integer k, and asks you if there is a vertex cover of size k. We can think of this as the language

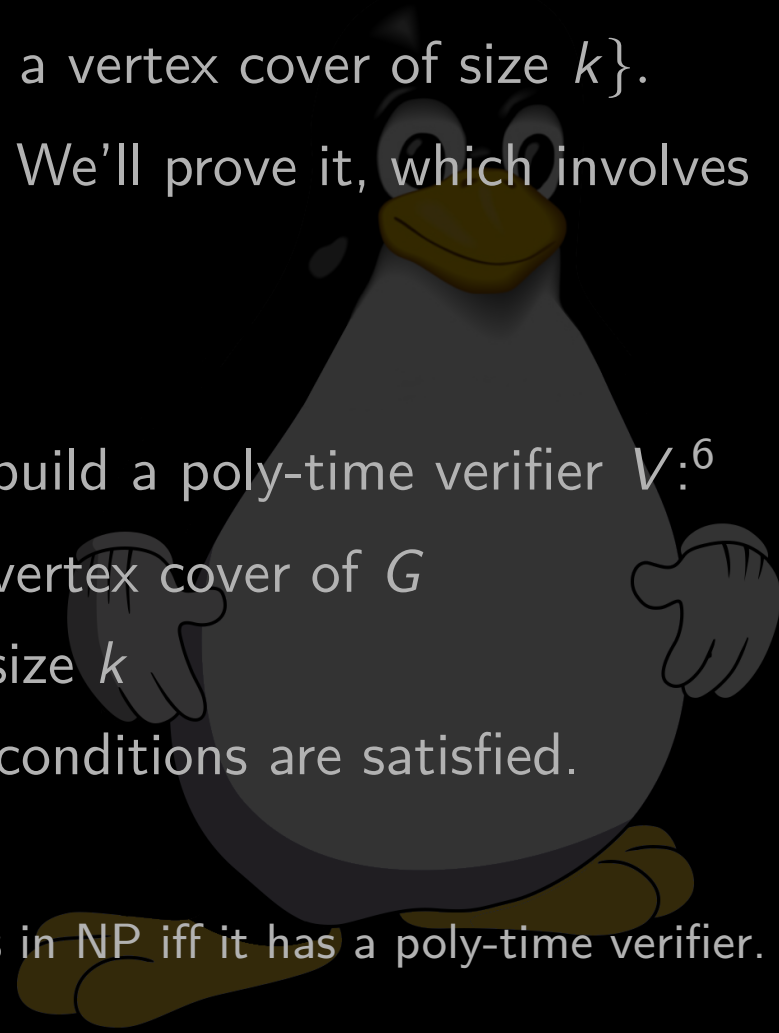
$$\text{VC} = \{(G, k) : G \text{ is a graph with a vertex cover of size } k\}.$$

Turns out this language is NP-complete! We'll prove it, which involves showing two things:

- $\text{VC} \in \text{NP}$;
- $A \leq_p \text{VC}$ for all languages $A \in \text{NP}$.

The first one is quick to prove: you can build a poly-time verifier V:[6]

$V(G, k, S)$: Check if S is a vertex cover of G
Check if S has size k
Accept iff both conditions are satisfied.



[6] Recall (from last tutorial) that a language is in NP iff it has a poly-time verifier. A verifier for a language A is a TM V such that

$$A = \{s : \text{there is a string } c \text{ such that } V(s, c) \text{ accepts}\}.$$

cover me owo 😳

The **vertex cover problem** gives you an arbitrary undirected graph G and an integer k, and asks you if there is a vertex cover of size k. We can think of this as the language

$$\text{VC} = \{(G, k) : G \text{ is a graph with a vertex cover of size } k\}.$$

Turns out this language is NP-complete! We'll prove it, which involves showing two things:

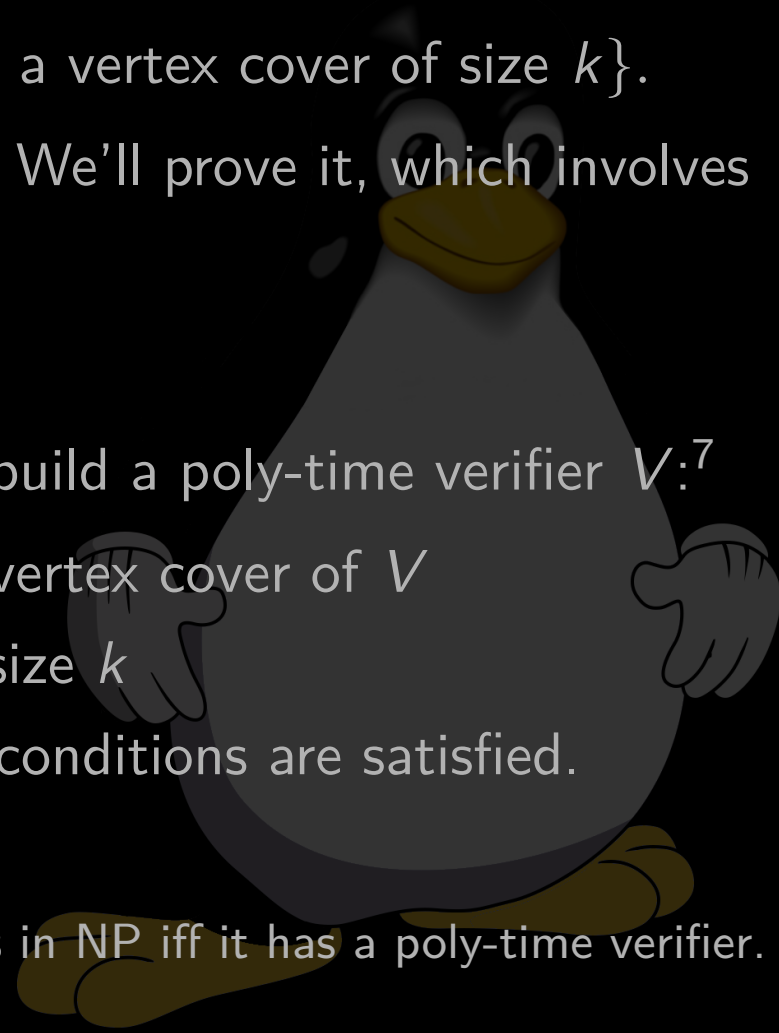
- $\text{VC} \in \text{NP}$;
- $A \leq_p \text{VC}$ for all languages $A \in \text{NP}$.

The first one is quick to prove: you can build a poly-time verifier V:[7]

$V(G, k, S)$: Check if S is a vertex cover of V

Check if S has size k

Accept iff both conditions are satisfied.



[7] Recall (from last tutorial) that a language is in NP iff it has a poly-time verifier. A verifier for a language A is a TM V such that

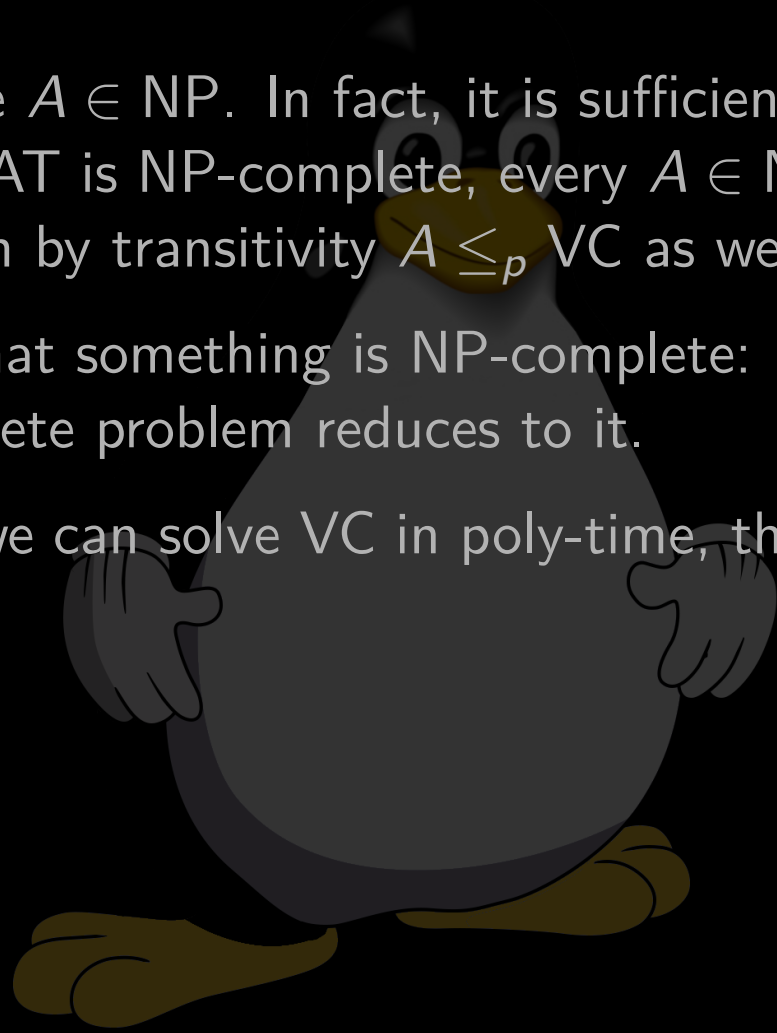
$$A = \{s : \text{there is a string } c \text{ such that } V(s, c) \text{ accepts}\}.$$

cover me owo 😳

We now prove $A \leq_p$ VC for any language $A \in$ NP. In fact, it is sufficient to show 3SAT $\leq_p$ VC: since we know 3SAT is NP-complete, every $A \in$ NP satisfies $A \leq_p$ 3SAT; if 3SAT $\leq_p$ VC then by transitivity $A \leq_p$ VC as well!

From now on this is how we will prove that something is NP-complete: just show some already known NP-complete problem reduces to it.

To show 3SAT $\leq_P$ VC, we show that if we can solve VC in poly-time, then we can solve 3SAT in poly-time as well.



3SAT $\leq_P$ VC 😳

Suppose we can solve VC in poly-time. Given a 3CNF formula φ, we will construct a graph G as follows:

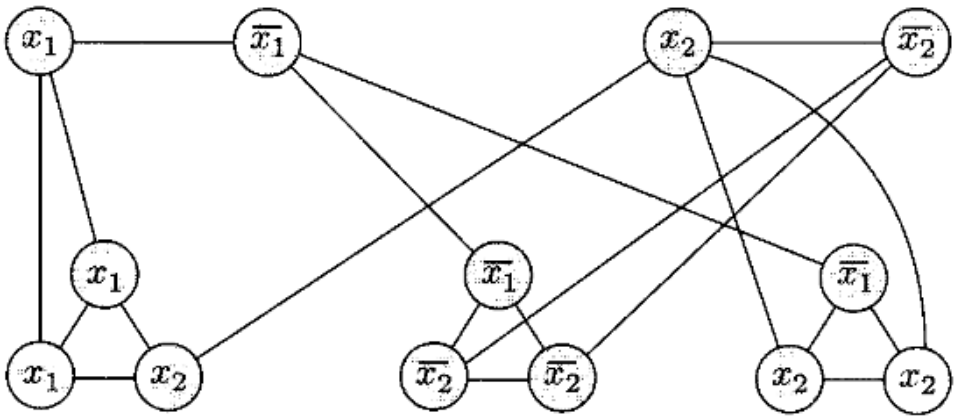
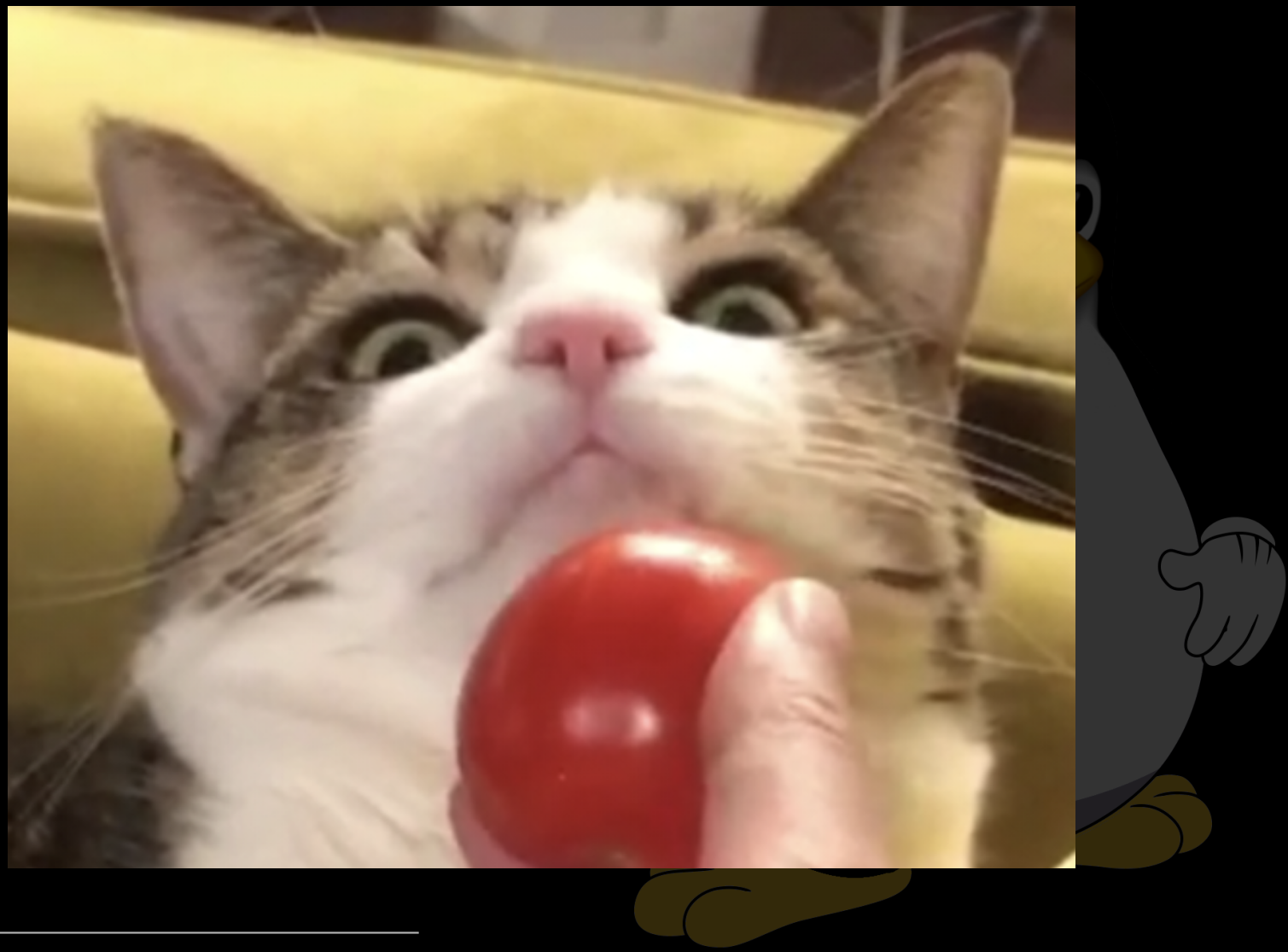


FIGURE 7.11
The graph that the reduction produces from
$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$

(uh, i'm probably just gonna narrate the rest of this! if you're just reading those slides, it's in Sipser's book on pg261)

bye! D: and hope today's tutorial[8] wasn't too flushed out.



[8] probably more like a lecture disguised as a review session.