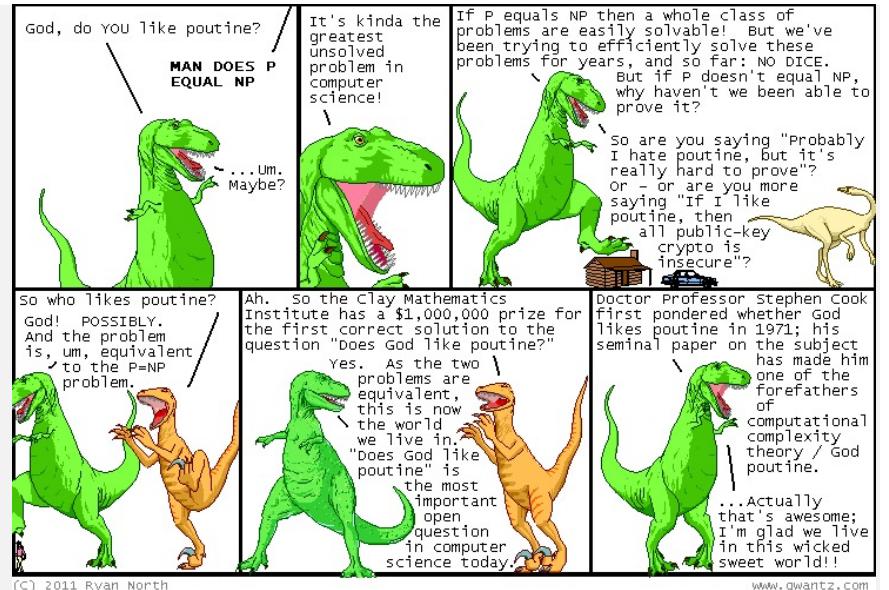
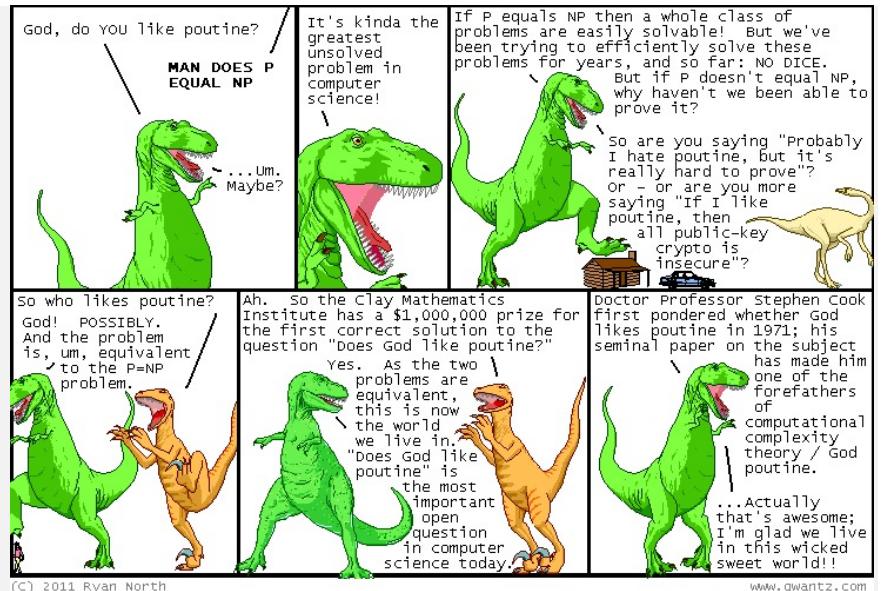


Poly Time Mapping Reducibility



Announcements



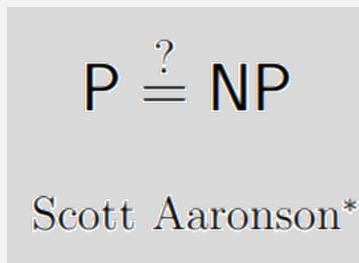
Last Time: P vs NP

- P = class of languages that can be decided “quickly”
 - i.e., “solvable” with a deterministic TM
- NP = class of languages that can be verified “quickly”
 - or, “solvable” with a nondeterministic TM
- Does P = NP ?
 - Problem first posed by John Nash
- It’s a difficult problem because how do you prove: “we’ll never find a poly time algorithm for X”?



Progress on whether $P = NP$?

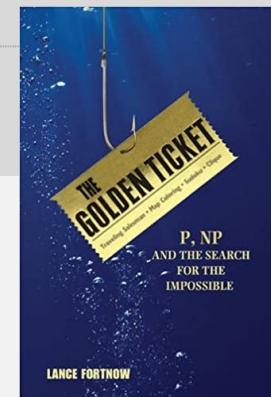
- Still not close



The Status of the P Versus NP Problem

By Lance Fortnow

Communications of the ACM, September 2009, Vol. 52 No. 9, Pages 78-86
10.1145/1562164.1562186



- One important concept discovered:
 - **NP**-Completeness (today)

Flashback: Mapping Reducibility

DEFINITION 5.20

Language A is **mapping reducible** to language B , written $A \leq_m B$, if there is a **computable function** $f: \Sigma^* \rightarrow \Sigma^*$, where for every w ,

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

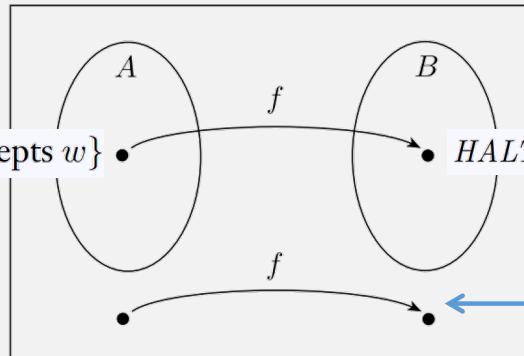
IMPORTANT: “if and only if” ...

The function f is called the **reduction** from A to B .

So to show mapping reducibility:

1. must create **computable fn**
2. and then show **forward direction**
3. and **reverse direction**

$$A_{\text{TM}} = \{\langle M, w \rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$$



... means $\overline{A} \leq_m \overline{B}$

DEFINITION 5.17

A function $f: \Sigma^* \rightarrow \Sigma^*$ is a **computable function** if some Turing machine M , on every input w , halts with just $f(w)$ on its tape.

Polynomial Time Mapping Reducibility

DEFINITION 5.20

Language A is *mapping reducible* to language B , written $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$, where for every w ,

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

The function f is called the *reduction* from A to B .

DEFINITION 7.29

Language A is *polynomial time mapping reducible*,¹ or simply *polynomial time reducible*, to language B , written $A \leq_P B$, if a polynomial time computable function $f: \Sigma^* \rightarrow \Sigma^*$ exists, where for every w ,

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

Don't Forget: "if and only if" ...

The function f is called the *polynomial time reduction* of A to B .

DEFINITION 5.17 **poly time** poly time
A function $f: \Sigma^* \rightarrow \Sigma^*$ is a *computable function* if some Turing machine M , on every input w , halts with just $f(w)$ on its tape.

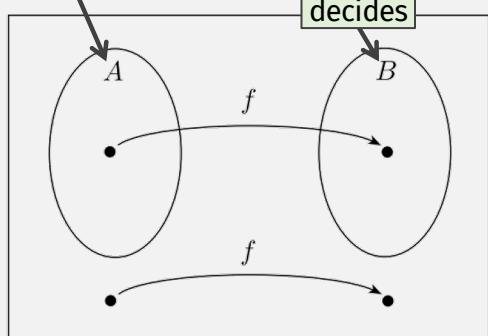
Flashback: If $A \leq_m B$ and B is decidable, then A is decidable.

(Theorem 5.22)

PROOF We let M be the decider for B and f be the reduction from A to B . We describe a decider N for A as follows.

N = “On input w :

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs.”



DEFINITION 5.20 —————

Language A is **mapping reducible** to language B , written $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$, where for every w ,

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

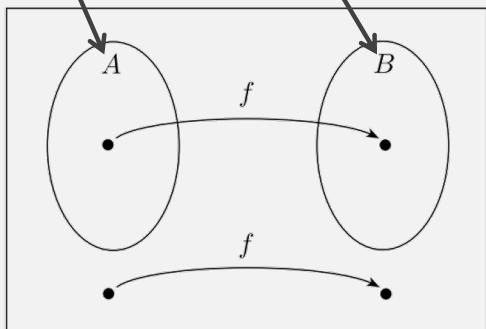
The function f is called the **reduction** from A to B .

THEOREM 7.31 If $A \leq_m B$ and $B \in P$, then $A \in P$.

PROOF We let M be the decider for B and f be the reduction from A to B . We describe a decider N for A as follows.

N = “On input w :

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs.”



DEFINITION 5.20

Language A is **mapping reducible** to language B , written $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$, where for every w ,

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

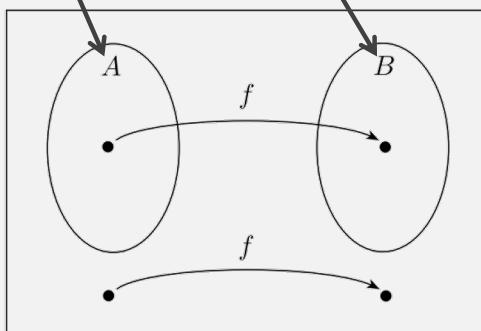
The function f is called the **reduction** from A to B .

THEOREM 7.31 If $A \leq_m B$ and $B \in P$, then $A \in P$.

PROOF We let M be the decider for B and f be the reduction from A to B .
We describe a decider N for A as follows.

N = “On input w :

1. Compute $f(w)$.
2. Run M on input $f(w)$ and output whatever M outputs.”



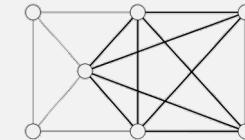
DEFINITION 5.20

Language A is **mapping reducible** to language B , written $A \leq_m B$, if there is a computable function $f: \Sigma^* \rightarrow \Sigma^*$, where for every w ,

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

The function f is called the **reduction** from A to B .

Theorem: $3SAT$ is polynomial time reducible to $CLIQUE$.



Last Class: ***CLIQUE*** is in NP

$\text{CLIQUE} = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

PROOF IDEA The clique is the certificate.

PROOF The following is a verifier V for *CLIQUE*.

V = “On input $\langle \langle G, k \rangle, c \rangle$:

1. Test whether c is a subgraph with k nodes in G . $O(k)$
2. Test whether G contains all edges connecting nodes in c . $O(k^2)$
3. If both pass, *accept*; otherwise, *reject*.”

DEFINITION 7.18 —

A *verifier* for a language A is an algorithm V , where

$$A = \{w \mid V \text{ accepts } \langle w, c \rangle \text{ for some string } c\}.$$

We measure the time of a verifier only in terms of the length of w , so a **polynomial time verifier** runs in polynomial time in the length of w . A language A is **polynomially verifiable** if it has a polynomial time verifier.

DEFINITION 7.19 —

NP is the class of languages that have polynomial time verifiers.

NP-Completeness

Must prove for
all langs, not
just one single
language

DEFINITION 7.34

A language B is **NP-complete** if it satisfies two conditions:

1. B is in NP, and **easy**
2. every A in NP is **polynomial time reducible** to B . **hard???? ...**

- How does this help the $P = NP$ problem?

THEOREM 7.35

If B is NP-complete and $B \in P$, then $P = NP$

... figuring out the
first NP-Complete
problem is hard!

(Just like figuring out the first
undecidable problem was hard!)

But then we use that problem to prove other problems NP-Complete!

Theorem: $3SAT$ is polynomial time reducible to $CLIQUE$.

??

Boolean Formulas

A Boolean _____	Is ...	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE

Boolean Formulas

A Boolean _____	Is ...	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean value	x, y, z

Boolean Formulas

A Boolean _____	Is ...	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean value	x, y, z
Operation	Combines Boolean variables	AND, OR, NOT (\wedge , \vee , and \neg)

Boolean Formulas

A Boolean _____	Is ...	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean value	x, y, z
Operation	Combines Boolean variables	AND, OR, NOT (\wedge , \vee , and \neg)
Formula ϕ	Combines vars and operations	$(\bar{x} \wedge y) \vee (x \wedge \bar{z})$

Boolean Satisfiability

- A Boolean formula is satisfiable if ...
- ... there is some assignment of TRUE or FALSE (1 or 0) to its variables that makes the entire formula TRUE
- Is $(\bar{x} \wedge y) \vee (x \wedge \bar{z})$ satisfiable?
 - Yes
 - $x = 0, y = 1, z = 0$

The Boolean Satisfiability Problem

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

Show SAT is in **NP**:

- Let n = the number of variables in the formula
- Verifier:
 - Let the certificate c be some assignment of variables to values
 - Verifying whether this assignment satisfies the formula takes time $O(n)$
- Non-deterministic Decider:
 - Non-deterministically try all possible assignments in parallel
 - Checking each assignment again takes time $O(n)$
- What about 3SAT?

More Boolean Formulas

A Boolean _____	Is ...	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean value	x, y, z
Operation	Combines Boolean variables	AND, OR, NOT (\wedge , \vee , and \neg)
Formula ϕ	Combines vars and operations	$(\bar{x} \wedge y) \vee (x \wedge \bar{z})$

More Boolean Formulas

A Boolean _____	Is ...	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean value	x, y, z
Operation	Combines Boolean variables	AND, OR, NOT (\wedge , \vee , and \neg)
Formula ϕ	Combines vars and operations	$(\bar{x} \wedge y) \vee (x \wedge \bar{z})$
Literal	A var or a negated var	x or \bar{x}

More Boolean Formulas

A Boolean _____	Is ...	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean value	x, y, z
Operation	Combines Boolean variables	AND, OR, NOT (\wedge , \vee , and \neg)
Formula ϕ	Combines vars and operations	$(\bar{x} \wedge y) \vee (x \wedge \bar{z})$
Literal	A var or a negated var	x or \bar{x}
Clause	Literals ORed together	$(x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4)$

More Boolean Formulas

A Boolean _____	Is ...	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean value	x, y, z
Operation	Combines Boolean variables	AND, OR, NOT (\wedge , \vee , and \neg)
Formula ϕ	Combines vars and operations	$(\bar{x} \wedge y) \vee (x \wedge \bar{z})$
Literal	A var or a negated var	x or \bar{x}
Clause	Literals ORed together	$(x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4)$
Conjunctive Normal Form (CNF)	Clauses ANDed together	$(x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4) \wedge (x_3 \vee \bar{x}_5 \vee x_6)$

More Boolean Formulas

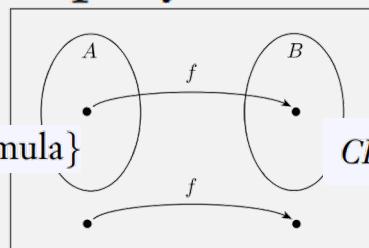
A Boolean _____	Is ...	Example:
Value	TRUE or FALSE (or 1 or 0)	TRUE, FALSE
Variable	Represents a Boolean value	x, y, z
Operation	Combines Boolean variables	AND, OR, NOT (\wedge , \vee , and \neg)
Formula ϕ	Combines vars and operations	$(\bar{x} \wedge y) \vee (x \wedge \bar{z})$
Literal	A var or a negated var	x or \bar{x}
Clause	Literals ORed together	$(x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4)$
Conjunctive Normal Form (CNF)	Clauses ANDed together	$(x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4) \wedge (x_3 \vee \bar{x}_5 \vee x_6)$
3CNF Formula	Three literals in each clause	$(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (x_3 \vee \bar{x}_5 \vee x_6) \wedge (x_3 \vee \bar{x}_6 \vee x_4)$

The $3SAT$ Problem

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

Theorem: $3SAT$ is polynomial time reducible to $CLIQUE$.

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



$CLIQUE = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$

Need: poly time computable fn converting a 3cnf-formula ...

$$\phi = (x_1 \vee x_1 \vee \boxed{x_2}) \wedge (\boxed{\bar{x}_1} \vee \bar{x}_2 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2 \vee \boxed{x_2})$$

- ... to a graph containing a clique:

- Each clause is a group of 3 nodes
- Connect all nodes except:
- Contradictory nodes

Don't forget iff
Nodes in the same group

=> If $\phi \in 3SAT$

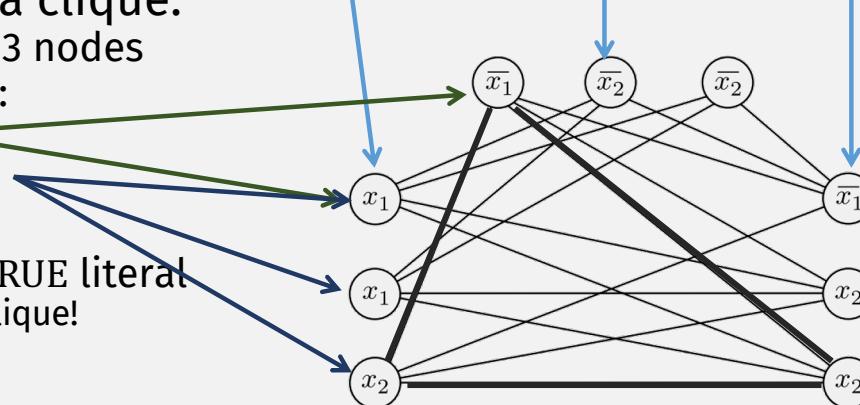
- Then each clause has a TRUE literal
- Those are nodes in the clique!
- eg $x_1 = 0, x_2 = 1$

<= If $f(\phi)$ in $CLIQUE$

- Each node of clique must come from different group
- So original formula is satisfiable, by because each group can be made TRUE

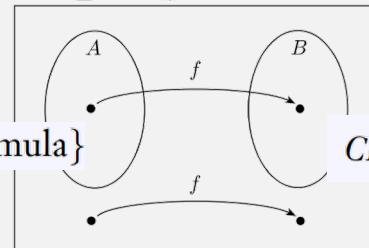
Must show fn runs in **polynomial time**:

- # literals = $O(k)$
- # nodes = $O(k)$
- # edges poly in # nodes $O(k^2)$



Theorem: $3SAT$ is polynomial time reducible to $CLIQUE$.

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



$$CLIQUE = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\}$$

- So since $CLIQUE$ is in **NP**, then $3SAT$ is also in **NP**

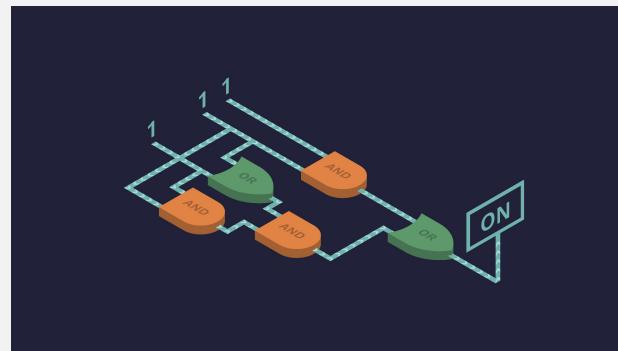
Next time: The Cook-Levin Theorem

The first NP-
Complete
problem

THEOREM 7.37

SAT is NP-complete

But it makes sense that every
problem can be reduced to it ...



Check-in
On gradescope