BILL AND NATHAN, RECORD LECTURE!!!!

BILL RECORD LECTURE!!!

NPC SAT-type Problems

Exposition by William Gasarch—U of MD

Theory of P and NP: Paradigm Shift

Exposition by William Gasarch—U of MD

Computability and Complexity

Computability The study of what problems can be solved and which ones cannot be solved. HALT is undecidable. Time is not an issue.

Computability and Complexity

Computability The study of what problems can be solved and which ones cannot be solved. HALT is undecidable. Time is not an issue.

Computability The study of what problems can be solved in good time and which ones cannot be solved in good time. We thing SAT cannot be solved in good time.

A Brief History of Theory of Complexity

Spoiler Alert The theory of computing made it possible to not just answer, but to ASK questions that had been around for a while.

A Brief History of Theory of Complexity

Spoiler Alert The theory of computing made it possible to not just answer, but to ASK questions that had been around for a while.

Hence the attention to time bounds is a **Paradigm Shift**: Old problems can be seen in a new light, and new problems can be stated.

A Brief History of Theory of Complexity

Spoiler Alert The theory of computing made it possible to not just answer, but to ASK questions that had been around for a while.

Hence the attention to time bounds is a **Paradigm Shift**: Old problems can be seen in a new light, and new problems can be stated.

I will present two threads of history of Theory of Computing.

A Brief History of Theory of Complexity

Spoiler Alert The theory of computing made it possible to not just answer, but to ASK questions that had been around for a while.

Hence the attention to time bounds is a **Paradigm Shift**: Old problems can be seen in a new light, and new problems can be stated.

I will present two threads of history of Theory of Computing.

Warning I am not a historian so some of what I say here may be exaggerated or wrong. But the general gist is correct.

Thread One: From Gauss to Gowers

Thread One: From Gauss to Gowers

1. In 1805 Gauss invented the Fast Fourier Transform for his own use and never thought to tell anyone. A statement like **FFT runs in $O(n \log n)$ time** would probably be very strange for him. In 1965 FFT was (re)invented by Cooley and Tukey.

Thread One: From Gauss to Gowers

1. In 1805 Gauss invented the Fast Fourier Transform for his own use and never thought to tell anyone. A statement like **FFT runs in $O(n \log n)$ time** would probably be very strange for him. In 1965 FFT was (re)invented by Cooley and Tukey.
2. In 1936 Turing defined **The Turing Machine** (he didn't call it that) as a model of computation. He **did not** concern himself with how many steps it took.

Thread One: From Gauss to Gowers

1. In 1805 Gauss invented the Fast Fourier Transform for his own use and never thought to tell anyone. A statement like **FFT runs in $O(n \log n)$ time** would probably be very strange for him. In 1965 FFT was (re)invented by Cooley and Tukey.
2. In 1936 Turing defined **The Turing Machine** (he didn't call it that) as a model of computation. He **did not** concern himself with how many steps it took.
3. During WW II Turing helped crack the German Enigma Code. This requires real computers solving problems quickly. Turing did not combine this with his other work. (Of course, he was busy winning WW II at the time.)

Thread One: From Gauss to Gowers

1. In 1805 Gauss invented the Fast Fourier Transform for his own use and never thought to tell anyone. A statement like **FFT runs in $O(n \log n)$ time** would probably be very strange for him. In 1965 FFT was (re)invented by Cooley and Tukey.
2. In 1936 Turing defined **The Turing Machine** (he didn't call it that) as a model of computation. He **did not** concern himself with how many steps it took.
3. During WW II Turing helped crack the German Enigma Code. This requires real computers solving problems quickly. Turing did not combine this with his other work. (Of course, he was busy winning WW II at the time.)

Thread continued on next slide.

Thread One: From Gauss to Rabin (cont)

Thread One: From Gauss to Rabin (cont)

1. In the 1950's computability theory was developed. There was some notion that maybe this could be applied to real computing, but this notion was somewhere between **aspirational** and **lip-service**.

Thread One: From Gauss to Rabin (cont)

1. In the 1950's computability theory was developed. There was some notion that maybe this could be applied to real computing, but this notion was somewhere between **aspirational** and **lip-service**.
2. In 1956 Joe Kruskal (Clyde's Uncle) invented Kruskal's Algorithm for Minimal Spanning Tree. The original paper did not mention time bounds at all, just had some vague allusions to this method being faster.

Thread One: From Gauss to Rabin (cont)

1. In the 1950's computability theory was developed. There was some notion that maybe this could be applied to real computing, but this notion was somewhere between **aspirational** and **lip-service**.
2. In 1956 Joe Kruskal (Clyde's Uncle) invented Kruskal's Algorithm for Minimal Spanning Tree. The original paper did not mention time bounds at all, just had some vague allusions to this method being faster.
3. In the early 1960's engineers and programmers began looking **informally** at how fast an algorithm takes to run.

A Thread One: From Gauss to Rabin (cont)

A Thread One: From Gauss to Rabin (cont)

1. In 1963 Hartmanis and Stearns define $\mathrm{DTIME}(T(n))$- a problem such that there is a Turing Machine that will, on inputs of length n, solve it in $\leq T(n)$ time.

A Thread One: From Gauss to Rabin (cont)

1. In 1963 Hartmanis and Stearns define $\mathrm{DTIME}(T(n))$- a problem such that there is a Turing Machine that will, on inputs of length n, solve it in $\le T(n)$ time.
2. In 1960's Knuth was a Math Ugrad by day and a programmer by night. He realized **Maybe I can use Math to analyze these algorithms!**

A Thread One: From Gauss to Rabin (cont)

1. In 1963 Hartmanis and Stearns define $\mathrm{DTIME}(T(n))$- a problem such that there is a Turing Machine that will, on inputs of length n, solve it in $\leq T(n)$ time.
2. In 1960's Knuth was a Math Ugrad by day and a programmer by night. He realized **Maybe I can use Math to analyze these algorithms!**
3. In 1965 Cobham defined polynomial time.

A Thread One: From Gauss to Rabin (cont)

1. In 1963 Hartmanis and Stearns define $\mathrm{DTIME}(T(n))$- a problem such that there is a Turing Machine that will, on inputs of length n, solve it in $\le T(n)$ time.
2. In 1960's Knuth was a Math Ugrad by day and a programmer by night. He realized **Maybe I can use Math to analyze these algorithms!**
3. In 1965 Cobham defined polynomial time.
4. Matching was known to be (in todays terms) $\mathrm{NP} \cap \mathrm{co\text{-}NP}$. In 1965 Jack Edmonds showed (in todays terms) that it was in P and defined P. He had ideas about NP and (in todays terms) conjectured $\mathrm{P} \ne \mathrm{NP}$. Possibly Whiggist History.

A Thread One: From Gauss to Rabin (cont)

1. In 1963 Hartmanis and Stearns define $\mathrm{DTIME}(T(n))$- a problem such that there is a Turing Machine that will, on inputs of length n, solve it in $\le T(n)$ time.
2. In 1960's Knuth was a Math Ugrad by day and a programmer by night. He realized **Maybe I can use Math to analyze these algorithms!**
3. In 1965 Cobham defined polynomial time.
4. Matching was known to be (in todays terms) $\mathrm{NP} \cap \text{co-NP}$. In 1965 Jack Edmonds showed (in todays terms) that it was in P and defined P. He had ideas about NP and (in todays terms) conjectured $\mathrm{P} \ne \mathrm{NP}$. Possibly Whiggist History.

Thread continued on next slide.

A Thread One: From Gauss to Rabin (cont)

A Thread One: From Gauss to Rabin (cont)

1. Cook (1971) and Levin (1973) define NP and prove SAT is NPC. Cook shows $3\mathrm{SAT}$ is NPC. Karp shows 21 problems are NPC. In the 1970's thousands of problems from different fields are shown NPC.

A Thread One: From Gauss to Rabin (cont)

1. Cook (1971) and Levin (1973) define NP and prove SAT is NPC. Cook shows $3\mathrm{SAT}$ is NPC. Karp shows 21 problems are NPC. In the 1970's thousands of problems from different fields are shown NPC.
2. In 1978 **The Handbook of Math Logic** had papers in many parts of logic.

A Thread One: From Gauss to Rabin (cont)

1. Cook (1971) and Levin (1973) define NP and prove SAT is NPC. Cook shows $3\mathrm{SAT}$ is NPC. Karp shows 21 problems are NPC. In the 1970's thousands of problems from different fields are shown NPC.
2. In 1978 **The Handbook of Math Logic** had papers in many parts of logic.**Only one** mentioned P vs NP:

A Thread One: From Gauss to Rabin (cont)

1. Cook (1971) and Levin (1973) define NP and prove SAT is NPC. Cook shows $\mathrm{3SAT}$ is NPC. Karp shows 21 problems are NPC. In the 1970's thousands of problems from different fields are shown NPC.
2. In 1978 **The Handbook of Math Logic** had papers in many parts of logic.**Only one** mentioned P vs NP: Rabin's paper on decidable theories noted that a theory could be decidable in theory but not in practice since it may be NP-hard. So the problem ws not well known in the Logic community and hence not well known in the math community since Logic would be its most natural place.

A Thread One: From Gauss to Rabin (cont)

1. Cook (1971) and Levin (1973) define NP and prove SAT is NPC. Cook shows $3\mathrm{SAT}$ is NPC. Karp shows 21 problems are NPC. In the 1970's thousands of problems from different fields are shown NPC.
2. In 1978 **The Handbook of Math Logic** had papers in many parts of logic.**Only one** mentioned P vs NP: Rabin's paper on decidable theories noted that a theory could be decidable in theory but not in practice since it may be NP-hard. So the problem ws not well known in the Logic community and hence not well known in the math community since Logic would be its most natural place.
3. In the 2000's Terry Tao and Timothy Gowers, two Field Medal winners, have tried to work on P vs NP. So the problem now has the respect of the Math community. Not sure if their working on is because the problem has respect or caused the problem to have respect.

Thread Two: Eulerian and Hamiltonian Graphs

Def

1. A graph is **Eulerian** if there is a cycle that hits every **edge** once.
2. A graph is **Hamiltonian** if there is a cycle that hits every **vertex** once.

Thread Two: Eulerian and Hamiltonian Graphs (cont)

1736 Euler solves the Konigsberg bridge problem by proving, in modern terms,
A graph is EUL iff every vertex has even degree. So $\mathrm{EUL} \in \mathrm{P}$.

Thread Two: Eulerian and Hamiltonian Graphs (cont)

1736 Euler solves the Konigsberg bridge problem by proving, in modern terms,
A graph is EUL iff every vertex has even degree. So $\mathrm{EUL} \in \mathrm{P}$.

1850? Hamilton poses, in modern terms, the question of characterizing when graphs are HAM.

Thread Two: Eulerian and Hamiltonian Graphs (cont)

1736 Euler solves the Konigsberg bridge problem by proving, in modern terms,
A graph is EUL iff every vertex has even degree. So $\mathrm{EUL} \in \mathrm{P}$.

1850? Hamilton poses, in modern terms, the question of characterizing when graphs are HAM.

Math Folks Wanted a similar **char of HAM graphs** but did not have a notion of algorithms so could not be rigorous.

Thread Two: Eulerian and Hamiltonian Graphs (cont)

1736 Euler solves the Konigsberg bridge problem by proving, in modern terms,
A graph is EUL iff every vertex has even degree. So $\mathrm{EUL} \in \mathrm{P}$.

1850? Hamilton poses, in modern terms, the question of characterizing when graphs are HAM.

Math Folks Wanted a similar **char of HAM graphs** but did not have a notion of algorithms so could not be rigorous.

NPC enabled people to **state** what they wanted $(\mathrm{HAM} \in \mathrm{P})$ and hence it could be shown unlikely $(\mathrm{HAM}$ is $\mathrm{NPC})$.

Thread Two: Eulerian and Hamiltonian Graphs (cont)

1736 Euler solves the Konigsberg bridge problem by proving, in modern terms,
A graph is EUL iff every vertex has even degree. So $\mathrm{EUL} \in \mathrm{P}$.

1850? Hamilton poses, in modern terms, the question of characterizing when graphs are HAM.

Math Folks Wanted a similar **char of HAM graphs** but did not have a notion of algorithms so could not be rigorous.

NPC enabled people to **state** what they wanted $(\mathrm{HAM} \in \mathrm{P})$ and hence it could be shown unlikely $(\mathrm{HAM}$ is $\mathrm{NPC})$.
Not an Isolated Example Many other vague open problems in math can now be stated more rigorously and either solved or shown hard to solve.

Why Do We Believe $P \neq NP$?

Exposition by William Gasarch—U of MD

Why Do We Believe $\mathrm{P} \neq \mathrm{NP}$?

Why Do We Believe $\mathrm{P} \neq \mathrm{NP}$?

1. There are 3 polls of what theorists think of P vs NP. 88% of those polled said $\mathrm{P} \neq \mathrm{NP}$.

Why Do We Believe $\mathrm{P} \neq \mathrm{NP}$?

1. There are 3 polls of what theorists think of P vs NP. 88% of those polled said $\mathrm{P} \neq \mathrm{NP}$. Some $\mathrm{P} = \mathrm{NP}$-ers emailed me privately that it was a protest vote—They think $\mathrm{P} \neq \mathrm{NP}$ but people should be more open minded.

Why Do We Believe $\mathrm{P} \neq \mathrm{NP}$?

1. There are 3 polls of what theorists think of P vs NP. 88% of those polled said $\mathrm{P} \neq \mathrm{NP}$. Some $\mathrm{P} = \mathrm{NP}$-ers emailed me privately that it was a protest vote—They think $\mathrm{P} \neq \mathrm{NP}$ but people should be more open minded. Knuth really does think $\mathrm{P} = \mathrm{NP}$ but that the NPC are in the hard part of P.

Why Do We Believe $\mathrm{P} \neq \mathrm{NP}$?

1. There are 3 polls of what theorists think of P vs NP. 88% of those polled said $\mathrm{P} \neq \mathrm{NP}$. Some $\mathrm{P} = \mathrm{NP}$-ers emailed me privately that it was a protest vote—They think $\mathrm{P} \neq \mathrm{NP}$ but people should be more open minded. Knuth really does think $\mathrm{P} = \mathrm{NP}$ but that the NPC are in the hard part of P.
2. The NPC problems have been worked on for a long time (many before P and NP were defined) and none are in P.

Why Do We Believe $\mathrm{P} \neq \mathrm{NP}$?

1. There are 3 polls of what theorists think of P vs NP. 88% of those polled said $\mathrm{P} \neq \mathrm{NP}$. Some $\mathrm{P} = \mathrm{NP}$-ers emailed me privately that it was a protest vote—They think $\mathrm{P} \neq \mathrm{NP}$ but people should be more open minded. Knuth really does think $\mathrm{P} = \mathrm{NP}$ but that the NPC are in the hard part of P.
2. The NPC problems have been worked on for a long time (many before P and NP were defined) and none are in P.
3. Intuition: **Coming up with a proof** seems harder than **Verifying a proof**.

Why Do We Believe $\mathrm{P} \neq \mathrm{NP}$?

1. There are 3 polls of what theorists think of P vs NP. 88% of those polled said $\mathrm{P} \neq \mathrm{NP}$. Some $\mathrm{P} = \mathrm{NP}$-ers emailed me privately that it was a protest vote—They think $\mathrm{P} \neq \mathrm{NP}$ but people should be more open minded. Knuth really does think $\mathrm{P} = \mathrm{NP}$ but that the NPC are in the hard part of P.
2. The NPC problems have been worked on for a long time (many before P and NP were defined) and none are in P.
3. Intuition: **Coming up with a proof** seems harder than **Verifying a proof**.
4. $\mathrm{P} \neq \mathrm{NP}$ has great explanatory power. See next slide.

Approximating Set Cover

Set Cover Given n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$ find the least number of sets S_i's that **cover** $\{1, \ldots, n\}$.

Approximating Set Cover

Set Cover Given n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$ find the least number of sets S_i's that **cover** $\{1, \ldots, n\}$.

1. Chvatal in 1979 showed that there is a poly time approx algorithm for **Set Cover** that will return $(\ln n) \times \mathrm{OPTIMAL}$.

Approximating Set Cover

Set Cover Given n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$ find the least number of sets S_i's that **cover** $\{1, \ldots, n\}$.

1. Chvatal in 1979 showed that there is a poly time approx algorithm for **Set Cover** that will return $(\ln n) \times \mathrm{OPTIMAL}$.
2. Dinur and Steurer in 2013 showed that, assuming $\mathrm{P} \neq \mathrm{NP}$, for all ϵ there is no $(1-\epsilon)\ln n \times \mathrm{OPTIMAL}$ approx alg for **Set Cover**

Approximating Set Cover

Set Cover Given n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$ find the least number of sets S_i's that **cover** $\{1, \ldots, n\}$.

1. Chvatal in 1979 showed that there is a poly time approx algorithm for **Set Cover** that will return $(\ln n) \times \mathrm{OPTIMAL}$.
2. Dinur and Steurer in 2013 showed that, assuming $\mathrm{P} \neq \mathrm{NP}$, for all ϵ there is no $(1-\epsilon)\ln n \times \mathrm{OPTIMAL}$ approx alg for **Set Cover**
3. These two proofs have nothing to do with each other yet give matching upper and lower bounds.

Approximating Set Cover

Set Cover Given n and $S_1, \ldots, S_m \subseteq \{1, \ldots, n\}$ find the least number of sets S_i's that **cover** $\{1, \ldots, n\}$.

1. Chvatal in 1979 showed that there is a poly time approx algorithm for **Set Cover** that will return $(\ln n) \times \mathrm{OPTIMAL}$.
2. Dinur and Steurer in 2013 showed that, assuming $\mathrm{P} \neq \mathrm{NP}$, for all ϵ there is no $(1-\epsilon) \ln n \times \mathrm{OPTIMAL}$ approx alg for **Set Cover**
3. These two proofs have nothing to do with each other yet give matching upper and lower bounds.
4. There are many other approx problems which (1) we have been unable to improve, and (2) $\mathrm{P} \neq \mathrm{NP}$ implies they cannot be improved.

NPC Problems on Boolean Formulas

Exposition by William Gasarch—U of MD

Bounding
(1) Literals Per Clause
(2) Occurrences of a Var

Exposition by William Gasarch—U of MD

Two Types of SAT

1. **kSAT-b**: Clauses have $\leq k$ literals, each var occurs $\leq b$ times.
2. **EU-kSAT-b**: Clauses have k literals, each var occurs $\leq b$ times.

Two Types of SAT

1. **kSAT-b**: Clauses have $\leq k$ literals, each var occurs $\leq b$ times.
2. **EU-kSAT-b**: Clauses have k literals, each var occurs $\leq b$ times.

Caveat Do not allow x and $\neg x$ in same clause.

Two Types of SAT

1. **kSAT-b**: Clauses have $\leq k$ literals, each var occurs $\leq b$ times.
2. **EU-kSAT-b**: Clauses have k literals, each var occurs $\leq b$ times.

Caveat Do not allow x and $\neg x$ in same clause.
Caveat Do not allow x and x in same clause.

Two Types of SAT

1. **kSAT-b**: Clauses have $\leq k$ literals, each var occurs $\leq b$ times.
2. **EU-kSAT-b**: Clauses have k literals, each var occurs $\leq b$ times.

Caveat Do not allow x and $\neg x$ in same clause.
Caveat Do not allow x and x in same clause.
Occur $(x \vee y) \wedge (\neg x \vee z)$: x occurs TWICE.

Two Types of SAT

1. **kSAT-b**: Clauses have $\le k$ literals, each var occurs $\le b$ times.
2. **EU-kSAT-b**: Clauses have k literals, each var occurs $\le b$ times.

Caveat Do not allow x and $\neg x$ in same clause.
Caveat Do not allow x and x in same clause.
Occur $(x \vee y) \wedge (\neg x \vee z)$: x occurs TWICE.
SAT means no bound on number of literals-per-clause.
We will look at all four of these for various values of k, b.

No Bound on b

1. 1SAT:

No Bound on *b*

1. $\mathrm{1SAT}$: P,
   $\phi \in \mathrm{1SAT}$ iff there is no x such that both x and $\neg x$ occur.
2. $\mathrm{2SAT}$:

No Bound on b

1. $\mathrm{1SAT}$: P,
   $\phi \in \mathrm{1SAT}$ iff there is no x such that both x and $\neg x$ occur.
2. $\mathrm{2SAT}$: P. Known result. Sketch: Convert every clause $L_1 \vee L_2$ into $(\neg L_1 \rightarrow L_2) \wedge (\neg L_2 \rightarrow L_1)$. Make a directed graph with literals as vertices and the $\rightarrow$ as edges. $\phi \in \mathrm{2SAT}$ iff there is no path from an x to a $\neg x$.
3. $\mathrm{3SAT}$: NPC by Cook.

The $k = 1$ and $k = 2$ cases are of course still in P if you bound b.

No Bound on b

1. $\mathrm{1SAT}$: P,
   $\phi \in \mathrm{1SAT}$ iff there is no x such that both x and $\neg x$ occur.
2. $\mathrm{2SAT}$: P. Known result. Sketch: Convert every clause $L_1 \vee L_2$ into $(\neg L_1 \rightarrow L_2) \wedge (\neg L_2 \rightarrow L_1)$. Make a directed graph with literals as vertices and the $\rightarrow$ as edges. $\phi \in \mathrm{2SAT}$ iff there is no path from an x to a $\neg x$.
3. $\mathrm{3SAT}$: NPC by Cook.

The $k = 1$ and $k = 2$ cases are of course still in P if you bound b. Hence we look at $k = 3$ and bound on b.

$k = 3$ and $b = 1, 2$

3SAT-1:

$k = 3$ and $b = 1, 2$

$\mathrm{3SAT\text{-}1}$: P. Always satisfiable, just set all literals that appear to T. EU version would still be in P.

$k = 3$ and $b = 1, 2$

3SAT-1: P. Always satisfiable, just set all literals that appear to T. EU version would still be in P.

3SAT-2: P? NPC? Work on in Breakout Rooms.

3SAT, all vars occur $\le$ 2. P

1) Input ϕ in 3CNF, all vars occurs ≤ 2.

$3\mathrm{SAT}$, all vars occur $\leq$ 2. P

1) Input ϕ in $3\mathrm{CNF}$, all vars occurs ≤ 2.
2) If a literal is only pos, set T, if only neg, set F. If clause has 1 literal, set true.
These operations may solve problem.

3SAT, all vars occur $\le$ 2. P

1) Input ϕ in $3\mathrm{CNF}$, all vars occurs ≤ 2.
2) If a literal is only pos, set T, if only neg, set F. If clause has 1 literal, set true.
These operations may solve problem.
3) Every clause has 2 or 3 literals, every literal occurs as pos and neg. We show SAT.

3SAT, all vars occur $\le$ 2. P

1) Input ϕ in $3\mathrm{CNF}$, all vars occurs ≤ 2.
2) If a literal is only pos, set T, if only neg, set F. If clause has 1 literal, set true.
These operations may solve problem.
3) Every clause has 2 or 3 literals, every literal occurs as pos and neg. We show SAT.
4) A clause with all NEG literals we call a NEG-clause.

3SAT, all vars occur $\leq$ 2. P

1) Input ϕ in $3\mathrm{CNF}$, all vars occurs ≤ 2.
2) If a literal is only pos, set T, if only neg, set F. If clause has 1 literal, set true.
These operations may solve problem.
3) Every clause has 2 or 3 literals, every literal occurs as pos and neg. We show SAT.
4) A clause with all NEG literals we call a NEG-clause.
If no NEG-clauses then SAT easily.

3SAT, all vars occur $\leq$ 2. P

1) Input ϕ in 3CNF, all vars occurs ≤ 2.
2) If a literal is only pos, set T, if only neg, set F. If clause has 1 literal, set true.
These operations may solve problem.
3) Every clause has 2 or 3 literals, every literal occurs as pos and neg. We show SAT.
4) A clause with all NEG literals we call a NEG-clause.
If no NEG-clauses then SAT easily.
IF there is a NEG-clause then set a var in it to F.

$\mathrm{3SAT}$, all vars occur ≤ 2. P

1) Input ϕ in $\mathrm{3CNF}$, all vars occurs ≤ 2.
2) If a literal is only pos, set T, if only neg, set F. If clause has 1 literal, set true.
These operations may solve problem.
3) Every clause has 2 or 3 literals, every literal occurs as pos and neg. We show SAT.
4) A clause with all NEG literals we call a NEG-clause.
If no NEG-clauses then SAT easily.
IF there is a NEG-clause then set a var in it to F.
(Numb NEG-clauses) + (Numb of clauses) DECREASES.

3SAT, all vars occur ≤ 2. P

1) Input ϕ in $3\mathrm{CNF}$, all vars occurs ≤ 2.
2) If a literal is only pos, set T, if only neg, set F. If clause has 1 literal, set true.
These operations may solve problem.
3) Every clause has 2 or 3 literals, every literal occurs as pos and neg. We show SAT.
4) A clause with all NEG literals we call a NEG-clause.
If no NEG-clauses then SAT easily.
IF there is a NEG-clause then set a var in it to F.
(Numb NEG-clauses) + (Numb of clauses) DECREASES.
Eventually satisfy all clauses.

$3SAT$, all vars occur ≤ 2. P

1) Input ϕ in $3CNF$, all vars occurs ≤ 2.
2) If a literal is only pos, set T, if only neg, set F. If clause has 1 literal, set true.
These operations may solve problem.
3) Every clause has 2 or 3 literals, every literal occurs as pos and neg. We show SAT.
4) A clause with all NEG literals we call a NEG-clause.
If no NEG-clauses then SAT easily.
IF there is a NEG-clause then set a var in it to F.
(Numb NEG-clauses) + (Numb of clauses) DECREASES.
Eventually satisfy all clauses.
Moral This was a clever trick! To prove $P \neq NP$ would need to show that no clever trick will get SAT into P. Hard!

3SAT, all vars occur $\leq$ 3

3SAT-3: There are ≤ 3 clauses per literal and every var occurs ≤ 3 times.

3SAT, all vars occur ≤ 3

3SAT-3: There are ≤ 3 clauses per literal and every var occurs ≤ 3 times.
In P? NPC? Breakout Rooms!

3SAT, all vars occur $\leq$ 3. NPC

We will prove this NPC. Erika- how will we do it?

3SAT, all vars occur ≤ 3. NPC

We will prove this NPC. Erika- how will we do it? By a Reduction
1) Input ϕ in 3CNF. Want ϕ' 3CNF with all vars occurring ≤ 3 times such that $\phi \in \mathrm{SAT}$ iff $\phi' \in \mathrm{SAT}$.

3SAT, all vars occur ≤ 3. NPC

We will prove this NPC. Erika- how will we do it? By a Reduction
1) Input ϕ in $3\mathrm{CNF}$. Want ϕ' $3\mathrm{CNF}$ with all vars occurring ≤ 3 times such that $\phi \in \mathrm{SAT}$ iff $\phi' \in \mathrm{SAT}$.
2) If a var occurs ≤ 3 times then leave it alone.

3SAT, all vars occur $\leq$ 3. NPC

We will prove this NPC. Erika- how will we do it? By a Reduction
1) Input ϕ in 3CNF. Want ϕ' 3CNF with all vars occurring ≤ 3 times such that $\phi \in \mathrm{SAT}$ iff $\phi' \in \mathrm{SAT}$.
2) If a var occurs ≤ 3 times then leave it alone.
3) If a var occurs $m \geq 4$ times then

3SAT, all vars occur ≤ 3. NPC

We will prove this NPC. Erika- how will we do it? By a Reduction
1) Input ϕ in 3CNF. Want ϕ' 3CNF with all vars occurring ≤ 3 times such that $\phi \in \mathrm{SAT}$ iff $\phi' \in \mathrm{SAT}$.
2) If a var occurs ≤ 3 times then leave it alone.
3) If a var occurs $m \ge 4$ times then
a) Add new vars $x_1, \ldots, x_m$. Replace ith occurrence of x with x_i.

3SAT, all vars occur ≤ 3. NPC

We will prove this NPC. Erika- how will we do it? By a Reduction
1) Input ϕ in 3CNF. Want ϕ' 3CNF with all vars occurring ≤ 3 times such that $\phi \in \mathrm{SAT}$ iff $\phi' \in \mathrm{SAT}$.
2) If a var occurs ≤ 3 times then leave it alone.
3) If a var occurs $m \ge 4$ times then
a) Add new vars $x_1, \ldots, x_m$. Replace ith occurrence of x with x_i.
b) Add the clauses $x_1 \rightarrow x_2$, $x_2 \rightarrow x_3$, …, $x_{m-1} \rightarrow x_m$, $x_m \rightarrow x_1$.
(Formally $x_1 \rightarrow x_2$ is $(\neg x_1 \vee x_2.)$)

3SAT, all vars occur ≤ 3. NPC

We will prove this NPC. Erika- how will we do it? By a Reduction
1) Input ϕ in 3CNF. Want ϕ' 3CNF with all vars occurring ≤ 3 times such that $\phi \in \text{SAT}$ iff $\phi' \in \text{SAT}$.
2) If a var occurs ≤ 3 times then leave it alone.
3) If a var occurs $m \geq 4$ times then
a) Add new vars $x_1, \ldots, x_m$. Replace ith occurrence of x with x_i.
b) Add the clauses $x_1 \rightarrow x_2$, $x_2 \rightarrow x_3$, …, $x_{m-1} \rightarrow x_m$, $x_m \rightarrow x_1$.
(Formally $x_1 \rightarrow x_2$ is $(\neg x_1 \vee x_2.)$
Clearly $\phi \in 3\text{CNF}$ and all variables occur ≤ 3 times.

3SAT, all vars occur ≤ 3. NPC

We will prove this NPC. Erika- how will we do it? By a Reduction
1) Input ϕ in 3CNF. Want ϕ' 3CNF with all vars occurring ≤ 3 times such that $\phi \in \mathrm{SAT}$ iff $\phi' \in \mathrm{SAT}$.
2) If a var occurs ≤ 3 times then leave it alone.
3) If a var occurs $m \ge 4$ times then
a) Add new vars $x_1, \ldots, x_m$. Replace ith occurrence of x with x_i.
b) Add the clauses $x_1 \rightarrow x_2$, $x_2 \rightarrow x_3$, . . ., $x_{m-1} \rightarrow x_m$, $x_m \rightarrow x_1$.
(Formally $x_1 \rightarrow x_2$ is $(\neg x_1 \vee x_2.)$
Clearly $\phi \in \mathrm{3CNF}$ and all variables occur ≤ 3 times.
Clearly $\phi \in \mathrm{SAT}$ iff $\phi' \in \mathrm{SAT}$

3SAT, all vars occur ≤ 3. NPC

We will prove this NPC. Erika- how will we do it? By a Reduction
1) Input ϕ in 3CNF. Want ϕ' 3CNF with all vars occurring ≤ 3 times such that $\phi \in \text{SAT}$ iff $\phi' \in \text{SAT}$.
2) If a var occurs ≤ 3 times then leave it alone.
3) If a var occurs $m \ge 4$ times then
a) Add new vars $x_1, \ldots, x_m$. Replace ith occurrence of x with x_i.
b) Add the clauses $x_1 \rightarrow x_2$, $x_2 \rightarrow x_3$, ..., $x_{m-1} \rightarrow x_m$, $x_m \rightarrow x_1$.
(Formally $x_1 \rightarrow x_2$ is $(\neg x_1 \vee x_2.)$)
Clearly $\phi \in \text{3CNF}$ and all variables occur ≤ 3 times.
Clearly $\phi \in \text{SAT}$ iff $\phi' \in \text{SAT}$
Moral Going from $b \le 2$ to $b \le 3$ matters!

EU-3SAT-3?

EU-3SAT-3: Every clause has **exactly 3** literals. Ever variable occurs ≤ 3 times. P? NPC?

EU-3SAT-3?

EU-3SAT-3: Every clause has **exactly 3** literals. Ever variable occurs ≤ 3 times. P? NPC?
Go to breakout rooms to work on this.

EU-3SAT-3 is in P

EU-3SAT-3 with $b \leq 3$ is in P.

EU-3SAT-3 is in P

EU-3SAT-3 with $b \le 3$ is in P.
This needs a known Theorem and its Corollary.
For this slide $G = (A, B, E)$ is a bipartite graph.
A **Matching of A into B** is a set of disjoint edges so that every element of A is an endpoint of some edge. View as an injection of A into B.
$X \subseteq A$. $E(X) = \{y \in Y : (\exists x \in X)[(x, y) \in E]\}$.

EU-3SAT-3 is in P

EU-3SAT-3 with $b \le 3$ is in P.
This needs a known Theorem and its Corollary.
For this slide $G = (A, B, E)$ is a bipartite graph.
A **Matching of A into B** is a set of disjoint edges so that every element of A is an endpoint of some edge. View as an injection of A into B.
$X \subseteq A$. $E(X) = \{y \in Y : (\exists x \in X)[(x, y) \in E]\}$.

Hall's Matching Theorem If, for all $X \subseteq A$, $|E(X)| \ge |X|$ then there exists a matching from A to B.

EU-3SAT-3 is in P

EU-3SAT-3 with $b \le 3$ is in P.
This needs a known Theorem and its Corollary.
For this slide $G = (A, B, E)$ is a bipartite graph.
A **Matching of A into B** is a set of disjoint edges so that every element of A is an endpoint of some edge. View as an injection of A into B.
$X \subseteq A$. $E(X) = \{y \in Y : (\exists x \in X)[(x, y) \in E]\}]$.

Hall's Matching Theorem If, for all $X \subseteq A$, $|E(X)| \ge |X|$ then there exists a matching from A to B.

Corollary If there exists k such that (1) for every $x \in A$, $\deg(x) \ge k$, and (2) for every $y \in B$, $\deg(y) \le k$, then there is a matching from A to B.

EU-3SAT-3 is in P

EU-3SAT-3 with $b \le 3$ is in P.
This needs a known Theorem and its Corollary.
For this slide $G = (A, B, E)$ is a bipartite graph.
A **Matching of A into B** is a set of disjoint edges so that every element of A is an endpoint of some edge. View as an injection of A into B.
$X \subseteq A$. $E(X) = \{y \in Y : (\exists x \in X)[(x, y) \in E]\}$.

Hall's Matching Theorem If, for all $X \subseteq A$, $|E(X)| \ge |X|$ then there exists a matching from A to B.

Corollary If there exists k such that (1) for every $x \in A$, $\deg(x) \ge k$, and (2) for every $y \in B$, $\deg(y) \le k$, then there is a matching from A to B.

We will use these on the next slide.

Every EU-3CNF-3 fml is Satisfiable

Let ϕ be EU-3CNF-3. $\phi = C_1 \vee \cdots \vee C_m$.
Form a bipartite graph:

1. Clauses on the left, variables on the right.
2. Edge from C to x if either x or $\neg x$ is in C.

Every clause has degree 3.

Every EU-3CNF-3 fml is Satisfiable

Let ϕ be EU-3CNF-3. $\phi = C_1 \vee \cdots \vee C_m$.
Form a bipartite graph:

1. Clauses on the left, variables on the right.
2. Edge from C to x if either x or $\neg x$ is in C.

Every clause has degree 3. Every variable has degree ≤ 3.
By Corollary there is a matching of C's to V's. This gives a satisfying assignment.

Every EU-3CNF-3 fml is Satisfiable

Let ϕ be EU-3CNF-3. $\phi = C_1 \vee \cdots \vee C_m$.
Form a bipartite graph:

1. Clauses on the left, variables on the right.
2. Edge from C to x if either x or $\neg x$ is in C.

Every clause has degree 3. Every variable has degree ≤ 3.
By Corollary there is a matching of C's to V's. This gives a satisfying assignment.
Moral The algorithm used a THEOREM in math that perhaps you did not know. To prove $\mathrm{P} \neq \mathrm{NP}$ would need to say this can't happen. Hard!

A Variant of SAT

Exposition by William Gasarch—U of MD

1-in-3-SAT

Def 1-in-3-SAT (1-in-3-SAT) is the problem of, given a formula $D_1 \wedge \cdots \wedge D_m$ find an assignment that satisfies **exactly** one literal-per-clause. We will show that 1-in-3-SAT is NPC.

1-in-3-SAT

Def 1-in-3-SAT (1-in-3-SAT) is the problem of, given a formula $D_1 \wedge \cdots \wedge D_m$ find an assignment that satisfies **exactly** one literal-per-clause. We will show that 1-in-3-SAT is NPC.
Is this a Natural Question? VOTE, though this is an opinion question.

1-in-3-SAT

Def **1-in-3-SAT (1-in-3-SAT)** is the problem of, given a formula $D_1 \wedge \cdots \wedge D_m$ find an assignment that satisfies **exactly** one literal-per-clause. We will show that 1-in-3-SAT is NPC.

Is this a Natural Question? VOTE, though this is an opinion question.

My Opinion The problem is **not** natural.

1-in-3-SAT

Def 1-in-3-SAT (1-in-3-SAT) is the problem of, given a formula $D_1 \wedge \cdots \wedge D_m$ find an assignment that satisfies **exactly** one literal-per-clause. We will show that 1-in-3-SAT is NPC.

Is this a Natural Question? VOTE, though this is an opinion question.

My Opinion The problem is **not** natural.

So why are we studying it Discuss.

1-in-3-SAT

Def 1-in-3-SAT (1-in-3-SAT) is the problem of, given a formula $D_1 \wedge \cdots \wedge D_m$ find an assignment that satisfies **exactly** one literal-per-clause. We will show that 1-in-3-SAT is NPC.

Is this a Natural Question? VOTE, though this is an opinion question.

My Opinion The problem is **not** natural.

So why are we studying it Discuss.

Its a means to an end We will show natural problems NPC by using reductions from 1-in-3-SAT. We will do a reduction from a variant of 1-in-3-SAT.

1-in-3-SAT is NPC

Given $\phi = C_1 \wedge \cdots \wedge C_m$ in $3\mathrm{CNF}$ create the ϕ' as follows:

1-in-3-SAT is NPC

Given $\phi = C_1 \wedge \cdots \wedge C_m$ in $3\mathrm{CNF}$ create the ϕ' as follows:
Replace clause $(L_1 \vee L_2 \vee L_3)$ with

$$(\neg L_1 \vee a \vee b) \wedge (b \vee L_2 \vee c) \wedge (c \vee d \vee \neg L_3).$$

where a, b, c, d are new variables.

1-in-3-SAT is NPC

Given $\phi = C_1 \wedge \cdots \wedge C_m$ in 3CNF create the ϕ' as follows:
Replace clause $(L_1 \vee L_2 \vee L_3)$ with

$$(\neg L_1 \vee a \vee b) \wedge (b \vee L_2 \vee c) \wedge (c \vee d \vee \neg L_3).$$

where a, b, c, d are new variables.
Leave it to the reader to prove

$$\phi \in \text{3SAT iff } \phi' \in \text{1-in-3-SAT}.$$

Mono 1-in-3-SAT

Mono 1-in-3-SAT (mono-1-in-3-SAT): Given a formula $E_1 \wedge \cdots \wedge E_p$ where all vars occur positively, is there an assignment that satisfies **exactly** one literal-per-clause.

Mono 1-in-3-SAT

Mono 1-in-3-SAT (mono-1-in-3-SAT): Given a formula $E_1 \wedge \cdots \wedge E_p$ where all vars occur positively, is there an assignment that satisfies **exactly** one literal-per-clause.

Thm $\text{1-in-3-SAT} \le \text{mono-1-in-3-SAT}$
Given 3CNF form $\phi(x_1, \ldots, x_n) = C_1 \vee \cdots \vee C_k$ want ϕ' such that $\phi \in \text{1-in-3-SAT}$ iff $\phi' \in \text{mono-1-in-3-SAT}$.

Mono 1-in-3-SAT

Mono 1-in-3-SAT (mono-1-in-3-SAT)**:** Given a formula $E_1 \wedge \cdots \wedge E_p$ where all vars occur positively, is there an assignment that satisfies **exactly** one literal-per-clause.

Thm $\text{1-in-3-SAT} \le \text{mono-1-in-3-SAT}$
Given 3CNF form $\phi(x_1, \ldots, x_n) = C_1 \vee \cdots \vee C_k$ want ϕ' such that $\phi \in \text{1-in-3-SAT}$ iff $\phi' \in \text{mono-1-in-3-SAT}$.
1) New Vars t, f and new clause $E = (t \vee f \vee f)$. Any 1-in-3-SAT assignment of ϕ will set t to T and f to F.

Mono 1-in-3-SAT

Mono 1-in-3-SAT (mono-1-in-3-SAT): Given a formula $E_1 \wedge \cdots \wedge E_p$ where all vars occur positively, is there an assignment that satisfies **exactly** one literal-per-clause.

Thm 1-in-3-SAT $\le$ mono-1-in-3-SAT

Given 3CNF form $\phi(x_1, \ldots, x_n) = C_1 \vee \cdots \vee C_k$ want ϕ' such that $\phi \in$ 1-in-3-SAT iff $\phi' \in$ mono-1-in-3-SAT.

1) New Vars t, f and new clause $E = (t \vee f \vee f)$. Any 1-in-3-SAT assignment of ϕ will set t to T and f to F.

2) For each x_j have new var x_j' and clause $D_j = (f \vee x_j \vee x_j')$. Any 1-in-3-SAT assignment for ϕ will set x_j, x_j' to opposites.

Mono 1-in-3-SAT

Mono 1-in-3-SAT (mono-1-in-3-SAT): Given a formula $E_1 \wedge \cdots \wedge E_p$ where all vars occur positively, is there an assignment that satisfies **exactly** one literal-per-clause.

Thm 1-in-3-SAT $\le$ mono-1-in-3-SAT
Given 3CNF form $\phi(x_1, \ldots, x_n) = C_1 \vee \cdots \vee C_k$ want ϕ' such that $\phi \in$ 1-in-3-SAT iff $\phi' \in$ mono-1-in-3-SAT.
1) New Vars t, f and new clause $E = (t \vee f \vee f)$. Any 1-in-3-SAT assignment of ϕ will set t to T and f to F.
2) For each x_j have new var x_j' and clause $D_j = (f \vee x_j \vee x_j')$. Any 1-in-3-SAT assignment for ϕ will set x_j, x_j' to opposites.
3) For each C_i let C_i' be obtained by replacing every $\overline{x_j}$ with x_j'.

Mono 1-in-3-SAT

Mono 1-in-3-SAT (mono-1-in-3-SAT): Given a formula $E_1 \wedge \cdots \wedge E_p$ where all vars occur positively, is there an assignment that satisfies **exactly** one literal-per-clause.

Thm 1-in-3-SAT $\le$ mono-1-in-3-SAT
Given 3CNF form $\phi(x_1, \ldots, x_n) = C_1 \vee \cdots \vee C_k$ want ϕ' such that $\phi \in$ 1-in-3-SAT iff $\phi' \in$ mono-1-in-3-SAT.
1) New Vars t, f and new clause $E = (t \vee f \vee f)$. Any 1-in-3-SAT assignment of ϕ will set t to T and f to F.
2) For each x_j have new var x_j' and clause $D_j = (f \vee x_j \vee x_j')$. Any 1-in-3-SAT assignment for ϕ will set x_j, x_j' to opposites.
3) For each C_i let C_i' be obtained by replacing every $\overline{x_j}$ with x_j'.

$$\phi' = C_1' \wedge \cdots \wedge C_k' \wedge D_1 \wedge \cdots \wedge D_n \wedge E.$$

Leave it to the reader to show $\phi \in$ 1-in-3-SAT iff $\phi' \in$ mono-1-in-3-SAT.

A Puzzle we Prove Hard Using mono-1-in-3-SAT

Exposition by William Gasarch—U of MD

Why is mono-1-in-3-SAT Important?

We care about the mono-1-in-3-SAT problem!

Why is mono-1-in-3-SAT Important?

We care about the mono-1-in-3-SAT problem! **NOT!**

Why is mono-1-in-3-SAT Important?

We care about the mono-1-in-3-SAT problem! **NOT!**

We will use it to show that a puzzle we DO care about is NPC

Why is mono-1-in-3-SAT Important?

We care about the mono-1-in-3-SAT problem! **NOT!**
We will use it to show that a puzzle we DO care about is NPC

	S	E	N	D
+	M	O	R	E
M	O	N	E	Y

The SEND MORE MONEY Cryptarithms

Why is mono-1-in-3-SAT Important?

We care about the mono-1-in-3-SAT problem! **NOT!**
We will use it to show that a puzzle we DO care about is NPC

	S	E	N	D
+	M	O	R	E
M	O	N	E	Y

The SEND MORE MONEY Cryptarithms

1) A carry can be at most 1. Hence $M = 1$.

Why is mono-1-in-3-SAT Important?

We care about the mono-1-in-3-SAT problem! **NOT!**
We will use it to show that a puzzle we DO care about is NPC

$$\begin{array}{ccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

The SEND MORE MONEY Cryptarithms

1) A carry can be at most 1. Hence $M = 1$.
2) Since $M = 1$, $S + M +$ poss carry ≤ 10. Since there is a carry, $S + M +$ poss carry $= 10$ so $O = 0$.

Why is mono-1-in-3-SAT Important?

We care about the mono-1-in-3-SAT problem! **NOT!**
We will use it to show that a puzzle we DO care about is NPC

$$\begin{array}{ccccc} & S & E & N & D \\ + & M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

The SEND MORE MONEY Cryptarithms

1) A carry can be at most 1. Hence $M = 1$.
2) Since $M = 1$, $S + M +$ poss carry ≤ 10. Since there is a carry, $S + M +$ poss carry $= 10$ so $O = 0$.
3) Can keep on reasoning like this and we find:

Why is mono-1-in-3-SAT Important?

We care about the mono-1-in-3-SAT problem! **NOT!**
We will use it to show that a puzzle we DO care about is NPC

	S	E	N	D
+	M	O	R	E
M	O	N	E	Y

The SEND MORE MONEY Cryptarithms

1) A carry can be at most 1. Hence $M = 1$.
2) Since $M = 1$, $S + M +$ poss carry ≤ 10. Since there is a carry, $S + M +$ poss carry $= 10$ so $O = 0$.
3) Can keep on reasoning like this and we find:

	9	5	6	7
+	1	0	8	5
1	0	6	5	2

The Solution to The SEND MORE MONEY Cryptarithms

How Did We Solve SEND+MORE=MONEY ?

We initially did some reasoning to cut down the number of poss.

How Did We Solve SEND+MORE=MONEY ?

We initially did some reasoning to cut down the number of poss.

But past a certain point we had to try all possibilities.

How Did We Solve SEND+MORE=MONEY ?

We initially did some reasoning to cut down the number of poss.

But past a certain point we had to try all possibilities.

Is the general problem NPC?

How Did We Solve SEND+MORE=MONEY ?

We initially did some reasoning to cut down the number of poss.

But past a certain point we had to try all possibilities.

Is the general problem NPC?
Spoiler Alert:

How Did We Solve SEND+MORE=MONEY ?

We initially did some reasoning to cut down the number of poss.

But past a certain point we had to try all possibilities.

Is the general problem NPC?
Spoiler Alert: **Yes**

Definition of Cryptarithms Problem

We want to show that Cryptarithms is NPC. We need a definition.

Definition of Cryptarithms Problem

We want to show that Cryptarithms is NPC. We need a definition.

CRYPTARITHM

Input $B, m \in \mathbb{N}$. Σ is alphabet of B letters.
$x_0, \ldots, x_{m-1}$. Each $x_i \in \Sigma$.
$y_0, \ldots, y_{m-1}$. Each $y_i \in \Sigma$.
$z_0, \ldots, z_m$. Each $z_i \in \Sigma$. The symbol z_m is optional.

Definition of Cryptarithms Problem

We want to show that Cryptarithms is NPC. We need a definition.

CRYPTARITHM

Input $B, m \in \mathbb{N}$. Σ is alphabet of B letters.
$x_0, \ldots, x_{m-1}$. Each $x_i \in \Sigma$.
$y_0, \ldots, y_{m-1}$. Each $y_i \in \Sigma$.
$z_0, \ldots, z_m$. Each $z_i \in \Sigma$. The symbol z_m is optional.
Question Does there exists injection $\Sigma \to \{0, \ldots, B-1\}$ so that the arithmetic below is correct in base B?

$$\begin{array}{cccc} & x_{m-1} & \cdots & x_0 \\ + & y_{m-1} & \cdots & y_0 \\ \hline z_m & z_{m-1} & \cdots & z_0 \end{array}$$

We Show CRYPTARITHM is NPC

Thm CRYPTARITHM is NPC.

We Show CRYPTARITHM is NPC

Thm CRYPTARITHM is NPC. Erika- How will we prove this?

We Show CRYPTARITHM is NPC

Thm CRYPTARITHM is NPC. Erika- How will we prove this?
We show mono-1-in-3-SAT $\le$ CRYPTARITHM. We show an algorithm that will:

We Show CRYPTARITHM is NPC

Thm CRYPTARITHM is NPC. Erika- How will we prove this?
We show mono-1-in-3-SAT $\le$ CRYPTARITHM. We show an algorithm that will:
Input $\phi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$ where all vars occur positive.

We Show CRYPTARITHM is NPC

Thm CRYPTARITHM is NPC. Erika- How will we prove this?
We show mono-1-in-3-SAT $\le$ CRYPTARITHM. We show an algorithm that will:
Input $\phi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$ where all vars occur positive.
Output An instance J of CRYPTARITHM such that TFAE

We Show CRYPTARITHM is NPC

Thm CRYPTARITHM is NPC. Erika- How will we prove this? We show mono-1-in-3-SAT $\le$ CRYPTARITHM. We show an algorithm that will:
Input $\phi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$ where all vars occur positive.
Output An instance J of CRYPTARITHM such that TFAE

1. Exists assignment that satisfies exactly one var per clause.
2. Exists solution to CRYPTARITHM J.

We Show CRYPTARITHM is NPC

Thm CRYPTARITHM is NPC. Erika- How will we prove this?
We show mono-1-in-3-SAT $\le$ CRYPTARITHM. We show an algorithm that will:
Input $\phi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$ where all vars occur positive.
Output An instance J of CRYPTARITHM such that TFAE

1. Exists assignment that satisfies exactly one var per clause.
2. Exists solution to CRYPTARITHM J.

We do the reduction in three parts, so three more slides!
We call the parts **gadgets**.

0 and 1

We have $0, 1 \in \Sigma$ that will live up their name.

0 and 1

We have $0, 1 \in \Sigma$ that will live up their name.
We have $p, q \in \Sigma$ that will help 0 maps to 0, 1 maps to 1.

0 and 1

We have $0, 1 \in \Sigma$ that will live up their name.
We have $p, q \in \Sigma$ that will help 0 maps to 0, 1 maps to 1.
We then make this part of J:

0 and 1

We have $0, 1 \in \Sigma$ that will live up their name.
We have $p, q \in \Sigma$ that will help 0 maps to 0, 1 maps to 1.
We then make this part of J:

$$\begin{array}{c} 0\,p\,0 \\ 0\,p\,0 \\ \hline 1\,q\,0 \end{array}$$

0 and 1

We have $0, 1 \in \Sigma$ that will live up their name.
We have $p, q \in \Sigma$ that will help 0 maps to 0, 1 maps to 1.
We then make this part of J:

$$\frac{\begin{matrix} 0\,p\,0 \\ 0\,p\,0 \end{matrix}}{1\,q\,0}$$

We leave it to the reader to show that this ensures 0 maps to 0 and 1 maps to 1.

Vars $\equiv 0, 1 \pmod 4$

For every variable v we have a symbol $v \in \Sigma$. Our intent is

Vars $\equiv 0, 1 \pmod 4$

For every variable v we have a symbol $v \in \Sigma$. Our intent is
If v is true then $v \equiv 1 \pmod 4$.

Vars $\equiv 0, 1 \pmod 4$

For every variable v we have a symbol $v \in \Sigma$. Our intent is
If v is true then $v \equiv 1 \pmod 4$.
If v is false then $v \equiv 0 \pmod 4$.

Vars $\equiv 0, 1 \pmod 4$

For every variable v we have a symbol $v \in \Sigma$. Our intent is
If v is true then $v \equiv 1 \pmod 4$.
If v is false then $v \equiv 0 \pmod 4$.
The following gadget ensures that $v \equiv 0, 1 \pmod 4$.

$$\begin{array}{cccccc} 0 & b & c & 0 & a & 0 \\ 0 & b & c & 0 & a & 0 \\ \hline 0 & v & d & 0 & b & 0 \end{array}$$

Vars $\equiv 0, 1 \pmod 4$

For every variable v we have a symbol $v \in \Sigma$. Our intent is
If v is true then $v \equiv 1 \pmod 4$.
If v is false then $v \equiv 0 \pmod 4$.
The following gadget ensures that $v \equiv 0, 1 \pmod 4$.

$$\begin{array}{cccccc} 0 & b & c & 0 & a & 0 \\ 0 & b & c & 0 & a & 0 \\ \hline 0 & v & d & 0 & b & 0 \end{array}$$

Since $a + a = b$ with no carry, $b \equiv 0 \pmod 2$.

Vars $\equiv 0,1 \pmod 4$

For every variable v we have a symbol $v \in \Sigma$. Our intent is
If v is true then $v \equiv 1 \pmod 4$.
If v is false then $v \equiv 0 \pmod 4$.
The following gadget ensures that $v \equiv 0,1 \pmod 4$.

$$\begin{array}{cccccc} 0 & b & c & 0 & a & 0 \\ 0 & b & c & 0 & a & 0 \\ \hline 0 & v & d & 0 & b & 0 \end{array}$$

Since $a + a = b$ with no carry, $b \equiv 0 \pmod 2$.

Since $c + c = d$ the carry is $C \in \{0, 1\}$.

Vars $\equiv 0,1 \pmod 4$

For every variable v we have a symbol $v \in \Sigma$. Our intent is
If v is true then $v \equiv 1 \pmod 4$.
If v is false then $v \equiv 0 \pmod 4$.
The following gadget ensures that $v \equiv 0,1 \pmod 4$.

$$
\begin{array}{cccccc}
0 & b & c & 0 & a & 0 \\
0 & b & c & 0 & a & 0 \\
\hline
0 & v & d & 0 & b & 0
\end{array}
$$

Since $a + a = b$ with no carry, $b \equiv 0 \pmod 2$.

Since $c + c = d$ the carry is $C \in \{0,1\}$.

Since $b + b = v$, $v = 2b + C$, so $v \equiv 0,1 \pmod 4$.

Vars $\equiv 0, 1 \pmod 4$

For every variable v we have a symbol $v \in \Sigma$. Our intent is
If v is true then $v \equiv 1 \pmod 4$.
If v is false then $v \equiv 0 \pmod 4$.
The following gadget ensures that $v \equiv 0, 1 \pmod 4$.

$$\begin{array}{cccccc} 0 & b & c & 0 & a & 0 \\ 0 & b & c & 0 & a & 0 \\ \hline 0 & v & d & 0 & b & 0 \end{array}$$

Since $a + a = b$ with no carry, $b \equiv 0 \pmod 2$.

Since $c + c = d$ the carry is $C \in \{0, 1\}$.

Since $b + b = v$, $v = 2b + C$, so $v \equiv 0, 1 \pmod 4$.

Note Do this for all vars v, using a different a, b, c for each one.

Clauses Need to have Exactly One Var True

Clause is $(x \vee y \vee z)$.

Clauses Need to have Exactly One Var True

Clause is $(x \vee y \vee z)$. Gadget is:

$$
\begin{array}{ccccccccccc}
0 & l & 0 & x & 0 & 1 & 0 & b & 0 & a & 0 \\
0 & z & 0 & y & 0 & c & 0 & b & 0 & a & 0 \\
\hline
0 & d & 0 & l & 0 & d & 0 & c & 0 & b & 0
\end{array}
$$

Clauses Need to have Exactly One Var True

Clause is $(x \vee y \vee z)$. Gadget is:

$$
\begin{array}{ccccccccccc}
0 & l & 0 & x & 0 & 1 & 0 & b & 0 & a & 0 \\
0 & z & 0 & y & 0 & c & 0 & b & 0 & a & 0 \\
\hline
0 & d & 0 & l & 0 & d & 0 & c & 0 & b & 0
\end{array}
$$

$a + a = b$, so $b \equiv 0 \pmod 2$.

Clauses Need to have Exactly One Var True

Clause is $(x \vee y \vee z)$. Gadget is:

$$\begin{array}{ccccccccccc} 0 & l & 0 & x & 0 & 1 & 0 & b & 0 & a & 0 \\ 0 & z & 0 & y & 0 & c & 0 & b & 0 & a & 0 \\ \hline 0 & d & 0 & l & 0 & d & 0 & c & 0 & b & 0 \end{array}$$

$a + a = b$, so $b \equiv 0 \pmod 2$.

$b + b = c$, so $c \equiv 0 \pmod 4$.

Clauses Need to have Exactly One Var True

Clause is $(x \vee y \vee z)$. Gadget is:

$$\begin{array}{ccccccccccc} 0 & l & 0 & x & 0 & 1 & 0 & b & 0 & a & 0 \\ 0 & z & 0 & y & 0 & c & 0 & b & 0 & a & 0 \\ \hline 0 & d & 0 & l & 0 & d & 0 & c & 0 & b & 0 \end{array}$$

$a + a = b$, so $b \equiv 0 \pmod 2$.

$b + b = c$, so $c \equiv 0 \pmod 4$.

$d = c + 1$ so $d \equiv 1 \pmod 4$.

Clauses Need to have Exactly One Var True

Clause is $(x \vee y \vee z)$. Gadget is:

$$
\begin{array}{ccccccccccc}
0 & l & 0 & x & 0 & 1 & 0 & b & 0 & a & 0 \\
0 & z & 0 & y & 0 & c & 0 & b & 0 & a & 0 \\
\hline
0 & d & 0 & l & 0 & d & 0 & c & 0 & b & 0
\end{array}
$$

$a + a = b$, so $b \equiv 0 \pmod 2$.

$b + b = c$, so $c \equiv 0 \pmod 4$.

$d = c + 1$ so $d \equiv 1 \pmod 4$.

$x + y = l$ so $x + y \equiv l \pmod 4$.

Clauses Need to have Exactly One Var True

Clause is $(x \vee y \vee z)$. Gadget is:

$$
\begin{array}{ccccccccccc}
0 & l & 0 & x & 0 & 1 & 0 & b & 0 & a & 0 \\
0 & z & 0 & y & 0 & c & 0 & b & 0 & a & 0 \\
\hline
0 & d & 0 & l & 0 & d & 0 & c & 0 & b & 0
\end{array}
$$

$a + a = b$, so $b \equiv 0 \pmod{2}$.

$b + b = c$, so $c \equiv 0 \pmod{4}$.

$d = c + 1$ so $d \equiv 1 \pmod{4}$.

$x + y = l$ so $x + y \equiv l \pmod{4}$.

$l + z = d$ so $x + y + z \equiv 1 \pmod{4}$.

Clauses Need to have Exactly One Var True

Clause is $(x \vee y \vee z)$. Gadget is:

0	l	0	x	0	1	0	b	0	a	0
0	z	0	y	0	c	0	b	0	a	0
0	d	0	l	0	d	0	c	0	b	0

$a + a = b$, so $b \equiv 0 \pmod 2$.

$b + b = c$, so $c \equiv 0 \pmod 4$.

$d = c + 1$ so $d \equiv 1 \pmod 4$.

$x + y = l$ so $x + y \equiv l \pmod 4$.

$l + z = d$ so $x + y + z \equiv 1 \pmod 4$.

Note For each clause use a different a, b, c, l.

Clauses Need to have Exactly One Var True

Clause is $(x \vee y \vee z)$. Gadget is:

0	l	0	x	0	1	0	b	0	a	0
0	z	0	y	0	c	0	b	0	a	0
0	d	0	l	0	d	0	c	0	b	0

$a + a = b$, so $b \equiv 0 \pmod 2$.

$b + b = c$, so $c \equiv 0 \pmod 4$.

$d = c + 1$ so $d \equiv 1 \pmod 4$.

$x + y = l$ so $x + y \equiv l \pmod 4$.

$l + z = d$ so $x + y + z \equiv 1 \pmod 4$.

Note For each clause use a different a, b, c, l.

So if J has a solution then ϕ has a 1-in-3 assignment.

Clauses Need to have Exactly One Var True

Clause is $(x \vee y \vee z)$. Gadget is:

$$\begin{array}{ccccccccccc} 0 & l & 0 & x & 0 & 1 & 0 & b & 0 & a & 0 \\ 0 & z & 0 & y & 0 & c & 0 & b & 0 & a & 0 \\ \hline 0 & d & 0 & l & 0 & d & 0 & c & 0 & b & 0 \end{array}$$

$a + a = b$, so $b \equiv 0 \pmod 2$.

$b + b = c$, so $c \equiv 0 \pmod 4$.

$d = c + 1$ so $d \equiv 1 \pmod 4$.

$x + y = l$ so $x + y \equiv l \pmod 4$.

$l + z = d$ so $x + y + z \equiv 1 \pmod 4$.

Note For each clause use a different a, b, c, l.

So if J has a solution then ϕ has a 1-in-3 assignment.
Need if ϕ has a 1-in-3 assignment then J has sol. Left to reader.