Big picture

- All languages
- Decidable

  Turing machines
- NP
- P
- Context-free

  Context-free grammars, push-down automata
- Regular

  Automata, non-deterministic automata,

  regular expressions

- Recall ATM =

  {(M,w) : M is a TM and M accepts w}

  is undecidable

- What about BTM =

  {(M,w) : M is a TM and M accepts w in ≤ 2^{500} steps} ?

- Is BTM undecidable?

- Recall ATM =

  {(M,w) : M is a TM and M accepts w}

  is undecidable

- What about BTM =

  {(M,w) : M is a TM and M accepts w in $\leq 2^{500}$ steps} ?

- BTM is decidable: Just run M on w for 2^{500} steps.
- Is this practical?

- Today computer: one instruction each 10^{-10} seconds
- Physical limit: one instruction each 10^{-43} seconds
- To run M for 2^{500} steps will always take
  $>> 10^{-43} \times 2^{500}$ seconds $>>$ 5 billion years
- The sun will die before then
- Conclusion: To run M for 2^{500} steps is impractical,
  regardless of hardware, programming language, etc.

- Complexity Theory studies which languages can be decided within a reasonable amount of time, and which languages cannot.

- How to measure time?
  Time of TM computation = number of TM steps

- We count steps as a function of the input length |w|
  Makes sense: need |w| steps just to read input w

Example: Recall the TM for $\{a^m b^m c^m : m \geq 0\}$:

M := “On input w:

(1) Scan tape and cross off one a, one b, and one c

(2) If none of these symbols is found, ACCEPT

(3) If not all of these symbols is found,

or if found in the wrong order, REJECT

(4) Go back to (1).”

How long does this take to run?

Example: Recall the TM for $\{a^m b^m c^m : m \geq 0\}$:

M := “On input w:

(1) Scan tape and cross off one a, one b, and one c

(2) If none of these symbols is found, ACCEPT

(3) If not all of these symbols is found,

or if found in the wrong order, REJECT

(4) Go back to (1).”

---

(1) takes 2*|w| steps (scan forward and back)

It is repeated at most ?? times

Example: Recall the TM for $\{a^m b^m c^m : m \geq 0\}$:

M := "On input w:

(1) Scan tape and cross off one a, one b, and one c

(2) If none of these symbols is found, ACCEPT

(3) If not all of these symbols is found,

or if found in the wrong order, REJECT

(4) Go back to (1)."

---

(1) takes $2^*|w|$ steps (scan forward and back)

It is repeated at most $|w|/3$ times (3 marks each time)

In total, the TM runs for at most ?? steps

Example: Recall the TM for $\{a^m b^m c^m : m \geq 0\}$:

M := "On input w:

(1) Scan tape and cross off one a, one b, and one c

(2) If none of these symbols is found, ACCEPT

(3) If not all of these symbols is found,

or if found in the wrong order, REJECT

(4) Go back to (1)."

---

(1) takes $2*|w|$ steps (scan forward and back)

It is repeated at most $|w|/3$ times (3 marks each time)

In total, the TM runs for at most $(2/3)*|w|^2$ steps.

Example: Recall the TM for $\{a^{2^m} : m \geq 0\}$:

M := “On input w,

(1) if only one a, ACCEPT

(2) cross off every other a on the tape

(3) if the number of a's is odd, REJECT

(4) Go back to 1)”

How long does this take to run?

Example: Recall the TM for $\{a^{2^m} : m \geq 0\}$:

M := "On input w,

(1) if only one a, ACCEPT

(2) cross off every other a on the tape

(3) if the number of a's is odd, REJECT

(4) Go back to 1)"

---

(2) takes 2*|w| steps (scan forward and back)

It is repeated at most ?? times

Example: Recall the TM for $\{a^{2^m} : m \geq 0\}$:

M := "On input w,

(1) if only one a, ACCEPT

(2) cross off every other a on the tape

(3) if the number of a's is odd, REJECT

(4) Go back to 1)"

---

(2) takes 2*|w| steps (scan forward and back)

It is repeated at most log(|w|) times, because each time half of remaining a's crossed off.

In total, the TM runs for at most ?? steps.

Example: Recall the TM for $\{a^{2^m} : m \geq 0\}$:

M := "On input w,

(1) if only one a, ACCEPT

(2) cross off every other a on the tape

(3) if the number of a's is odd, REJECT

(4) Go back to 1)"

---

(2) takes 2*|w| steps (scan forward and back)

It is repeated at most log(|w|) times, because each time half of remaining a's crossed off.

In total, the TM runs for at most 2*|w|*log(|w|) steps.

- Notation: Letter “n” usually stands for input length |w|

- Definition: Let $t : \mathbb{N} \rightarrow \mathbb{N}$ be a function

  TIME(t(n)) = { L : L can be decided by a TM

  that runs for at most t(n) steps

  on every input of length n}

- Example: $\{a^m b^m c^m : m \geq 0\} \in$ TIME($(2/3)n^2$)

  $\{a^{2^m} : m \geq 0\} \in$ TIME(2n log(n))

- How robust is this notion of time?

- Recall

- Theorem: For every language L :

  L decidable in JAVA ⇔ L decidable in TM

- Does anything like this hold for TIME?

- The time equivalence between JAVA, TM, and all other programming languages is not exact.

- There are languages that

  can be recognized in time n in JAVA,

  but require at least time n^2 on TM

- But surprisingly the gap is not much bigger than that:

- Theorem:

  There is an integer c such that, for every function t(n)

  TIME(t(n)) in JAVA $\subseteq$ TIME($t(n)^c$) on TM

  TIME($t(n)^c$) in JAVA $\supseteq$ TIME(t(n)) on TM

- Example:

  L $\in$ TIME(n) in JAVA $\Rightarrow$ L $\in$ TIME(??) on TM

  L $\in$ TIME(n^2) in JAVA $\Rightarrow$ L $\in$ TIME(??) on TM

- Small values, like c = 3, are possible

- Theorem:

  There is an integer c such that, for every function t(n)

  TIME(t(n)) in JAVA $\subseteq$ TIME($t(n)^c$) on TM

  TIME($t(n)^c$) in JAVA $\supseteq$ TIME(t(n)) on TM

- Example:

  $L \in$ TIME(n) in JAVA $\Rightarrow L \in$ TIME(n^c) on TM

  $L \in$ TIME(n^2) in JAVA $\Rightarrow L \in$ TIME(n^{2c}) on TM

- Small values, like c = 3, are possible

- Definition: Polynomial Time:

  $P := U_c \text{ TIME}(n^c) = \text{TIME}(n^1) \text{ U TIME}(n^2) \text{ U} \ldots$

- This class is invariant under computational model:

  P on JAVA is the same as P on TM

- Approximates intuitive notion of “efficient”

  As close as we get to model your laptop

  Most (all?) what you'll ever program is in P

- Previous examples: $\{a^m b^m c^m : m \geq 0\} \in P$

  $\{a^{2^m} : m \geq 0\} \in P$

- Definition: Polynomial Time:

  $P := \cup_c \text{TIME}(n^c) = \text{TIME}(n^1) \cup \text{TIME}(n^2) \cup \ldots$

- The Algorithms class studies languages in P

  There, you also distinguish between time n^2 and n^3

  For this distinction TM not fine enough

- This class studies what is NOT in P

  We do not distinguish between time n^2 and n^3

  We can work with TM

- What languages are not in P ?

- What languages are not in P ?

- Recall ATM:={(M,w) | M is a TM and M accepts w}
  We proved ATM undecidable, so ATM $\notin$ P.

- Despite intense research,
  ATM is essentially the only language
  we can prove to be outside of P

- Many other languages are believed to be not in P: SAT, factoring, etc.
- Among these, there is a class of interesting languages called NP-complete
- These include problems people care about solving, because they occur frequently in practice
- If any one of these problems is in P, then all are!

- Next: Define several NP-complete problems:
  SAT, CLIQUE, SUBSET-SUM, ...

- Prove polynomial-time reductions:

  CLIQUE $\in$ P $\Rightarrow$ SAT $\in$ P

  SUBSET-SUM $\in$ P $\Rightarrow$ SAT $\in$ P

- Definition: "A reduces to B in polynomial time" means:

  B $\in$ P $\Rightarrow$ A $\in$ P

- Conceptually like L decidable $\Rightarrow$ ATM decidable

- Definition of boolean formulas

(boolean) variable take either true or false (1 or 0)

literal = variable or its negation $x, \neg x$

clause = OR of literals $(x \vee \neg y \vee z)$

CNF = AND of clauses $(x \vee \neg y \vee z) \wedge (z) \wedge (\neg x \vee y)$

3CNF = CNF where each clause has 3 literals

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

A 3CNF is satisfiable if $\exists$ assignment of 1 or 0 to variables that make the formula true

Satisfying assignment for above 3CNF?

- Definition of boolean formulas

(boolean) variable take either true or false (1 or 0)

literal = variable or its negation $x, \neg x$

clause = OR of literals $(x \lor \neg y \lor z)$

CNF = AND of clauses $(x \lor \neg y \lor z) \land (z) \land (\neg x \lor y)$

3CNF = CNF where each clause has 3 literals

$$(x \lor \neg y \lor z) \land (z \lor y \lor w) \land (\neg x \lor y \lor \neg u)$$

A 3CNF is satisfiable if $\exists$ assignment of 1 or 0 to variables that make the formula true

$x = 1, y = 1$ satisfies above

Equivalently, assignment makes each clause true

- Definition 3SAT := { φ | φ is a satisfiable 3CNF}

- Example: (x V y V z) Λ (z V ¬y V ¬x) ?? 3SAT:

- Definition 3SAT := { φ | φ is a satisfiable 3CNF}

- Example: (x V y V z) Λ (z V ¬y V ¬x) ∈3SAT:

  Assignment x = 1, y = 0, z = 0 gives

  (1 V 0 V 0) Λ (0 V 1 V 0) = 1 Λ 1 = 1

  (x V x V x) Λ (¬x V ¬x V ¬x) ?? 3SAT

- Definition 3SAT := { φ | φ is a satisfiable 3CNF}

- Example: (x V y V z) Λ (z V ¬y V ¬x) ∈ 3SAT:

  Assignment x = 1, y = 0, z = 0 gives

  (1 V 0 V 0) Λ (0 V 1 V 0) = 1 Λ 1 = 1

  (x V x V x) Λ (¬x V ¬x V ¬x) ∉ 3SAT

  x = 0 gives 0 Λ 1 = 0, x = 1 gives 1 Λ 0 = 0

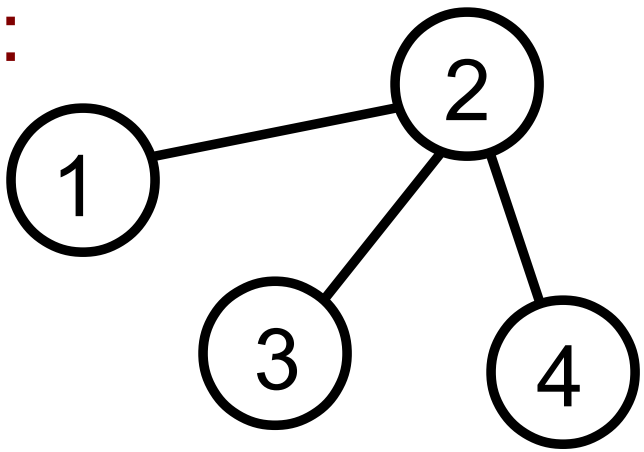
- Conjecture: 3SAT ∉ P
- Best known algorithm takes time exponential in | φ |

- Definition: a graph G = (V, E) consists of

  a set of nodes V (also called "vertices")

  a set of edges E that connect pairs of nodes

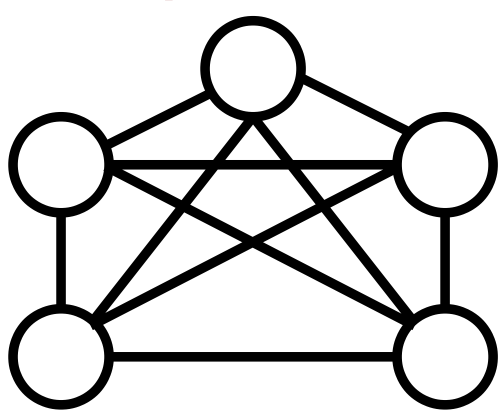
- Example:



V = {1, 2, 3, 4}

E = {(1,2), (2,3), (2,4)}

- Definition: a t-clique is a set of t nodes all connected
- Example:

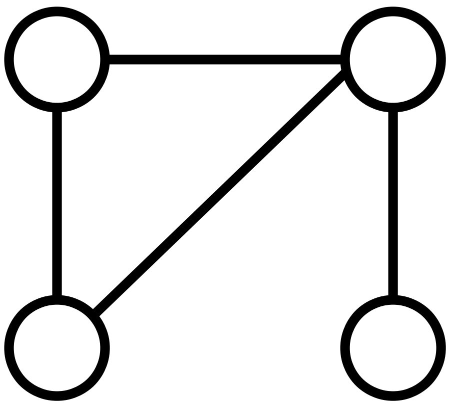


is a 5-clique

- Definition:

CLIQUE = {(G,t) : G is a graph containing a t-clique}

- Example:

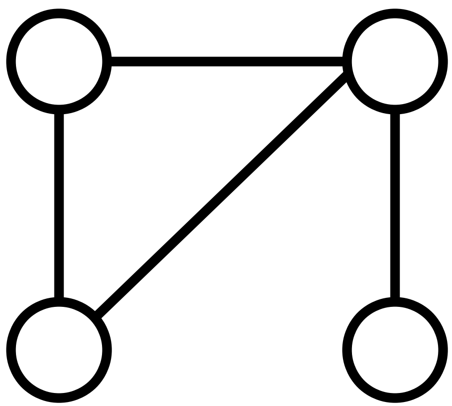
G = 

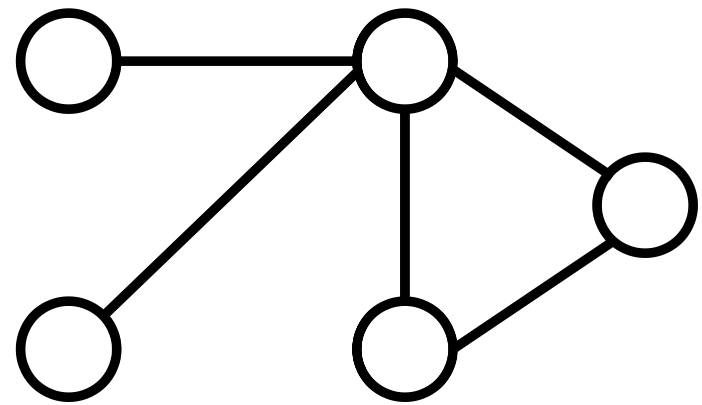
(G, 3) ? CLIQUE

- Definition:

CLIQUE = {(G,t) : G is a graph containing a t-clique}

- Example:

G = 

H = 

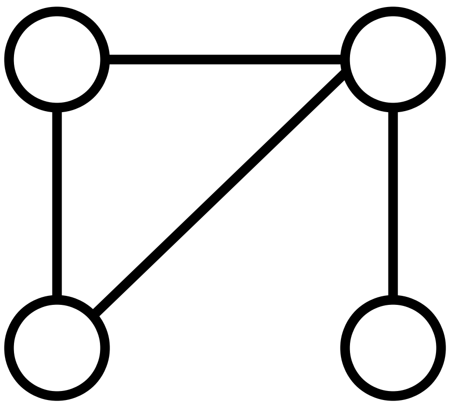
(G, 3) $\in$ CLIQUE

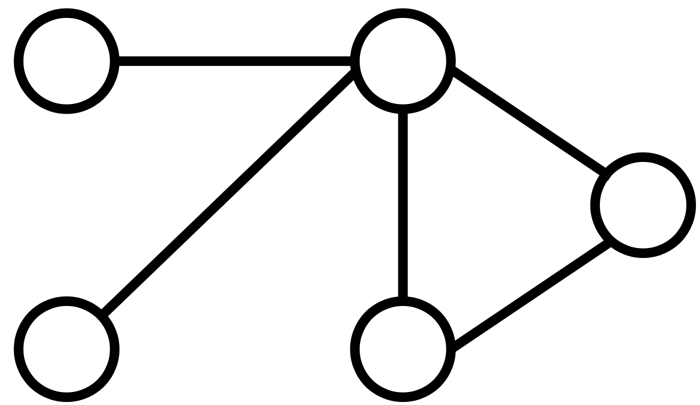
(H, 4) ? CLIQUE

- Definition:

CLIQUE = {(G,t) : G is a graph containing a t-clique}

- Example:

G = 

H = 

(G, 3) $\in$ CLIQUE

(H, 4) $\notin$ CLIQUE

- Conjecture: CLIQUE $\notin$ P

- 3SAT and CLIQUE both believed $\notin$ P
- They seem different problems. And yet:
- Theorem: CLIQUE $\in$ P $\Rightarrow$ 3SAT $\in$ P
- If you think 3SAT $\notin$ P, you also think CLIQUE $\notin$ P
- Above theorem gives what reduction?

- 3SAT and CLIQUE both believed $\notin$ P
- They seem different problems. And yet:
- Theorem: CLIQUE $\in$ P $\Rightarrow$ 3SAT $\in$ P
- If you think 3SAT $\notin$ P, you also think CLIQUE $\notin$ P
- Above theorem gives
  polynomial-time reduction of 3SAT to CLIQUE

- Theorem: CLIQUE $\in$ P $\Rightarrow$ 3SAT $\in$ P

- Proof outline:

We give TM R that on input φ :

(1) Computes graph G_φ and integer t_φ such that

$$\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$$

(2) R runs in polynomial time

Enough to prove the theorem?

- Theorem: CLIQUE $\in$ P $\Rightarrow$ 3SAT $\in$ P

- Proof outline:

We give TM R that on input φ :

(1) Computes graph G_φ and integer t_φ such that

$$\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$$

(2) R runs in polynomial time

Enough to prove the theorem because:

If $\exists$ TM C that solves CLIQUE in polynomial-time

Then C(R(φ)) solves 3SAT in polynomial-time

- Definition of R:

"On input

$$\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$$

Note a_i b_i c_i are literals, φ has k clauses

- Compute G_φ and t_φ as follows:
- Nodes of G_φ : one for each a_i , b_i , c_i
- Edges of G_φ : Connect all nodes except

(A) Nodes in same clause

(B) Contradictory nodes, such as x and $\neg$ x

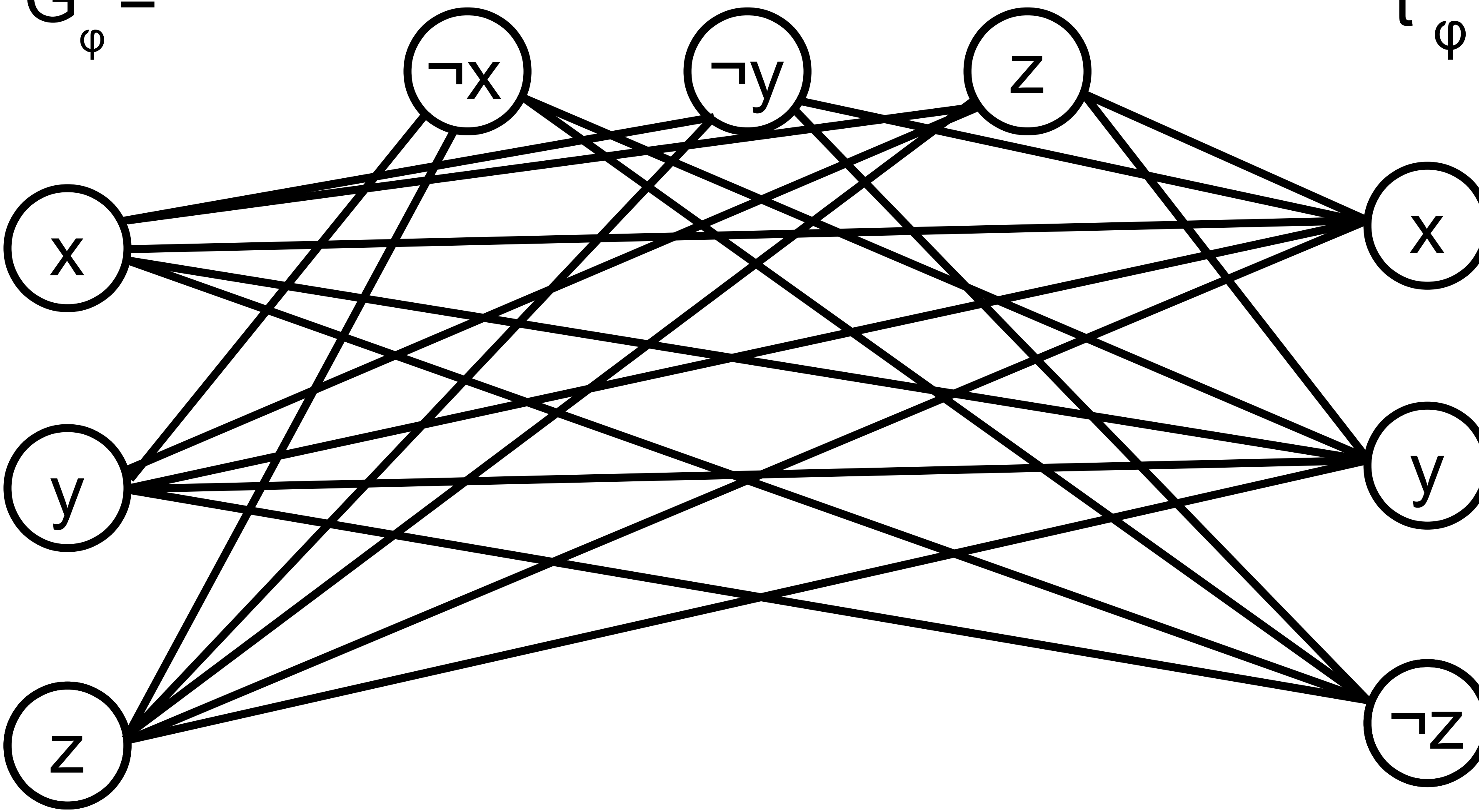
- t_φ := k"

Example:

$$\varphi = (x \lor y \lor z) \land (\neg x \lor \neg y \lor z) \land (x \lor y \lor \neg z)$$

$G_{\varphi} =$

$t_{\varphi} = 3$



- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$

- High-level view of proof of ⇨

We suppose φ has a satisfying assignment,

and we show a clique of size t_φ in G_φ

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$
- Proof: $\Rightarrow$

  Suppose φ has satisfying assignment
- So each clause must have at least one true literal
- Pick corresponding nodes in G_φ
- There are ??? nodes

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$
- Proof: ⇨

Suppose φ has satisfying assignment

- So each clause must have at least one true literal
- Pick corresponding nodes in G_φ
- There are $k = t_\varphi$ nodes
- They are a clique because in G_φ we connect all but

(A) Nodes in same clause

???

(B) Contradictory nodes.

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$
- Proof: $\Rightarrow$

Suppose φ has satisfying assignment

- So each clause must have at least one true literal
- Pick corresponding nodes in G_φ
- There are $k = t_\varphi$ nodes
- They are a clique because in G_φ we connect all but

(A) Nodes in same clause

Our nodes are picked from different clauses

(B) Contradictory nodes. ???

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi, t_\varphi) \in CLIQUE$
- Proof: $\Rightarrow$

Suppose φ has satisfying assignment

- So each clause must have at least one true literal
- Pick corresponding nodes in G_φ
- There are $k = t_\varphi$ nodes
- They are a clique because in G_φ we connect all but

(A) Nodes in same clause

Our nodes are picked from different clauses

(B) Contradictory nodes. Our nodes correspond to true literals in assignment: if x true then ¬ x can't be

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$

- High-level view of proof of $\Leftarrow$

- We suppose G_φ has a clique of size t_φ,

- then we show a satisfying assignment for φ

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$
- Proof: ⇦
- Suppose G_φ has a clique of size t_φ

- Note you have exactly one node per clause

  because ???

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$
- Proof: ⇦
- Suppose G_φ has a clique of size t_φ

- Note you have exactly one node per clause

  because by (A) there are no edges within clauses

- Define assignment that makes those literals true

  Possible ???

- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$
- Proof: ⇦
- Suppose G_φ has a clique of size t_φ

- Note you have exactly one node per clause

  because by (A) there are no edges within clauses

- Define assignment that makes those literals true

  Possible by (B): contradictory literals not connected

- Assignment satisfies φ because ???

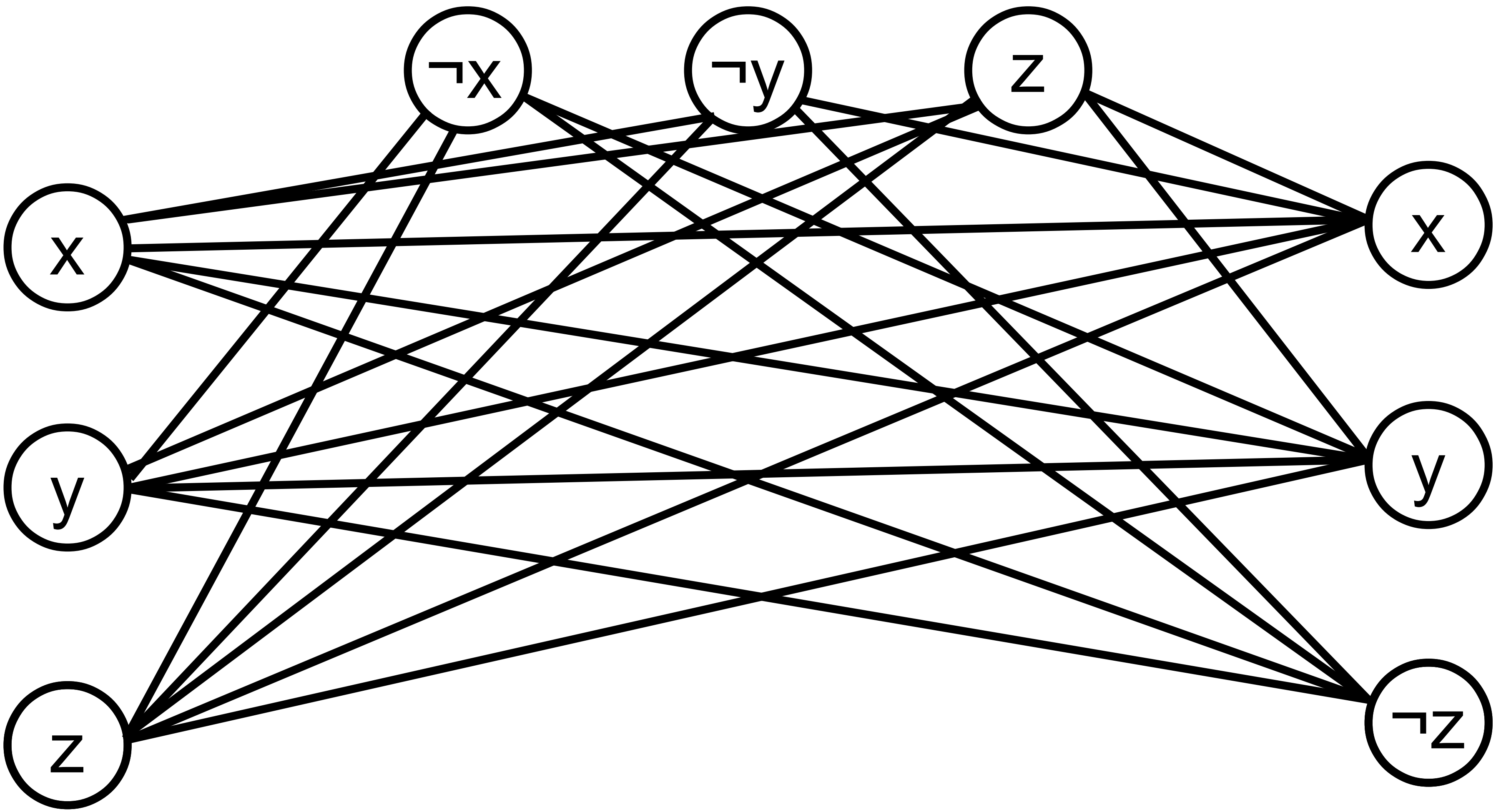
- Claim: $\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$
- Proof: ⇦
- Suppose G_φ has a clique of size t_φ
- Note you have exactly one node per clause

  because by (A) there are no edges within clauses
- Define assignment that makes those literals true

  Possible by (B): contradictory literals not connected
- Assignment satisfies φ because every clause is true

Back to example:

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

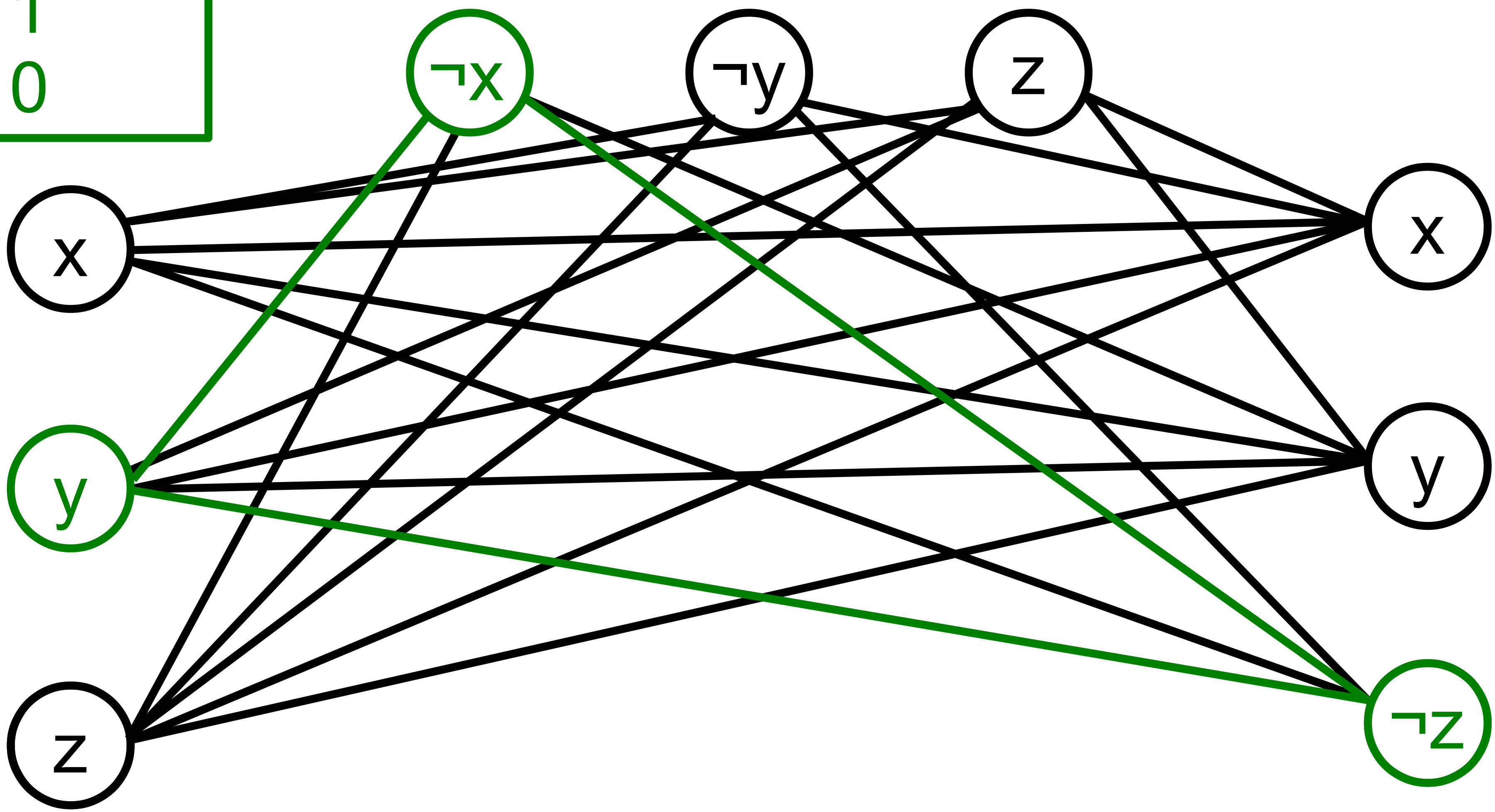


Back to example:

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

0 1 0 1 0 0 0 1 1

Assignment
x = 0
y = 1
z = 0



Back to example:

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

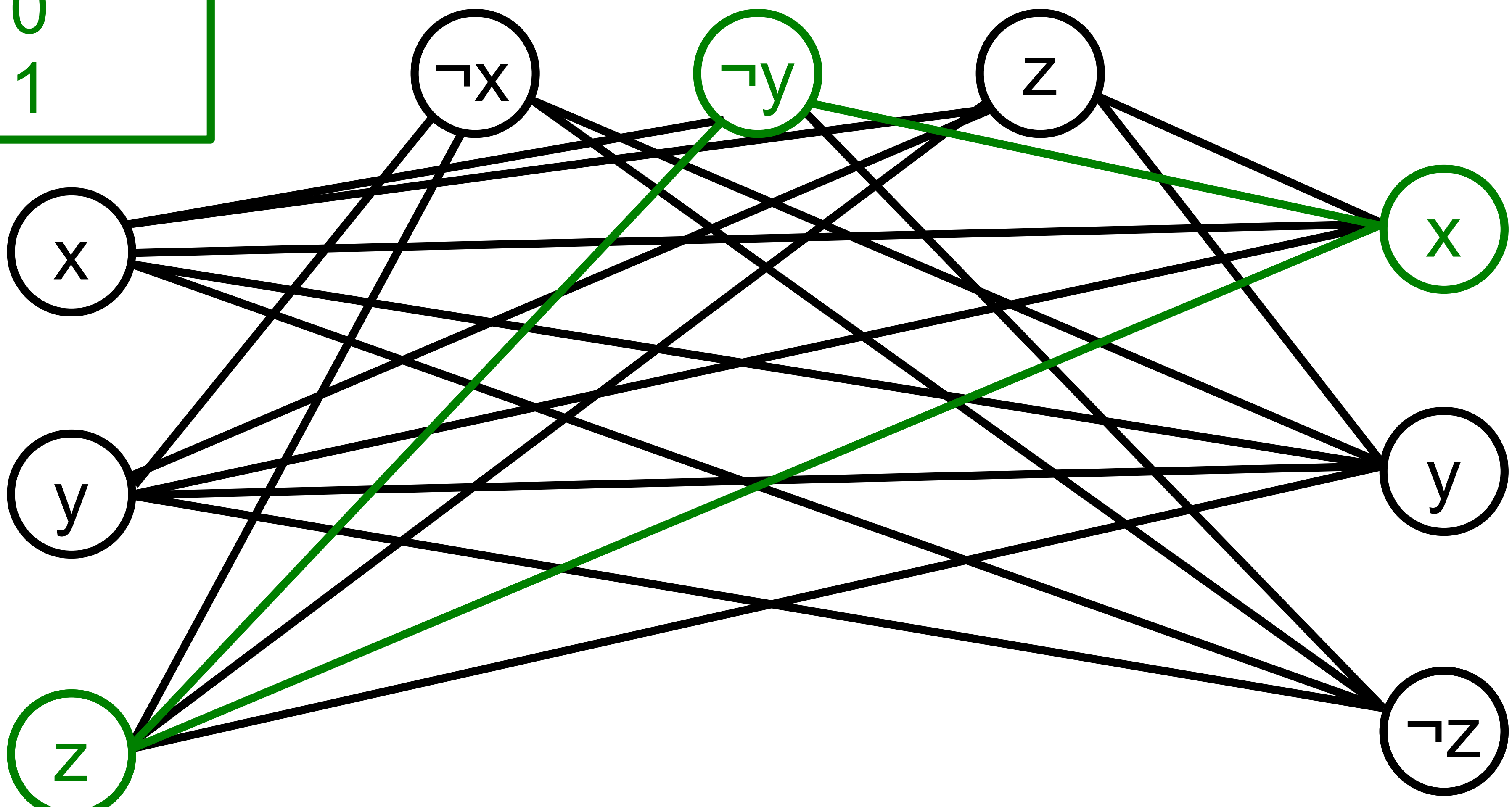
1 0 1   0 1 1   1 0 0

Assignment

x = 1

y = 0

z = 1



- Theorem: CLIQUE $\in$ P $\Rightarrow$ 3SAT $\in$ P

- Proof outline:

We give TM R that on input φ :

(1) Computes graph G_φ and integer t_φ such that

$$\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$$

(2) R runs in polynomial time

- So far: defined R, proved (1). It remains to see (2)

- (2) is less interesting.

- R : "On input $\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$

  Nodes of G_φ : one for each a_i b_i c_i

  Edges of G_φ : Connect all nodes except

  (A) Nodes in same clause

  (B) Contradictory nodes, such as x and ¬ x

  $t_\varphi := k$"

- We do not directly count the steps of TM R

  Too low-level, complicated, uninformative.

- We give a more high-level argument

- R : “On input $\varphi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge \ldots \wedge (a_k \vee b_k \vee c_k)$

  Nodes of G_φ : one for each a_i b_i c_i

  Edges of G_φ : Connect all nodes except

  (A) Nodes in same clause

  (B) Contradictory nodes, such as x and ¬ x

  $t_\varphi := k$”

- To compute nodes: examine all literals.

  Number of literals ≤ | φ |

- This is polynomial in the input length | φ |

- R : "On input $\varphi = (a_1 V b_1 V c_1) \wedge (a_2 V b_2 V c_2) \wedge \ldots \wedge (a_k V b_k V c_k)$
  Nodes of G_φ : one for each a_i b_i c_i
  Edges of G_φ : Connect all nodes except
  (A) Nodes in same clause
  (B) Contradictory nodes, such as x and ¬ x
  t_φ := k"

- To compute edges: examine all pairs of nodes.
  Number of pairs is ≤ (number of nodes)^2 ≤ $|\varphi|^2$

- Which is polynomial in the input length $|\varphi|$

- R : "On input $\varphi = (a_1 V b_1 V c_1) \Lambda (a_2 V b_2 V c_2) \Lambda \ldots \Lambda (a_k V b_k V c_k)$

  Nodes of G_φ : one for each a_i b_i c_i

  Edges of G_φ : Connect all nodes except

  (A) Nodes in same clause

  (B) Contradictory nodes, such as x and ¬ x

  $t_\varphi := k$"

- Overall, we examine $\leq | \varphi | + | \varphi |^2$

- Which is polynomial in the input length $| \varphi |$
- This concludes the proof.

- Theorem: CLIQUE $\in$ P $\Rightarrow$ 3SAT $\in$ P

- We have concluded the proof of above theorem

- Recall outline:

We give TM R that on input φ :

(1) Computes graph G_φ and integer t_φ such that

$$\varphi \in 3SAT \Leftrightarrow (G_\varphi , t_\varphi) \in CLIQUE$$

(2) R runs in polynomial time

- Definition:

SUBSET-SUM = $\{(a_1, a_2, ..., a_n, t) : \exists\ i1, i2, ..., ik \leq n$ such that $a_{i1}+a_{i2}+....+a_{ik} = t\ \}$

- Example:
  - (5, 2, 14, 3, 9, 25) ? SUBSET-SUM

- Definition:

SUBSET-SUM = $\{(a_1, a_2, ..., a_n, t) : \exists\ i1, i2, ..., ik \leq n$ such that $a_{i1}+a_{i2}+....+a_{ik} = t\ \}$

- Example:
  - (5, 2, 14, 3, 9, 25) $\in$ SUBSET-SUM

    because 2 + 14 + 9 = 25
  - (1, 3, 4, 9, 15) ? SUBSET-SUM

- Definition:

SUBSET-SUM = $\{(a_1, a_2, ..., a_n, t) : \exists\ i1, i2, ..., ik \leq n$ such that $a_{i1}+a_{i2}+....+a_{ik} = t\ \}$

- Example:
  - (5, 2, 14, 3, 9, 25) $\in$ SUBSET-SUM

    because 2 + 14 + 9 = 25
  - (1, 3, 4, 9, 15) $\notin$ SUBSET-SUM

    because no subset of {1,3,4,9} sums to 15

- Conjecture: SUBSET-SUM $\notin$ P

- **Theorem**: SUBSET-SUM $\in$ P $\Rightarrow$ 3SAT $\in$ P

- **Proof outline**:

We give TM R that on input φ :

(1) Computes numbers $a_1, a_2, \ldots, a_n, t$ such that

$$\varphi \in 3SAT \Leftrightarrow (a_1, a_2, \ldots, a_n, t) \in \text{SUBSET-SUM}$$

(2) R runs in polynomial time

- Theorem: SUBSET-SUM $\in$ P $\Rightarrow$ 3SAT $\in$ P

- Warm-up for definition of R:
- On input φ with v variables and k clauses:

- R will produce a list of numbers.
- Numbers will have many digits, v + k

  and look like this: 1000010011010011

  First v (most significant) digits correspond to variables
- Other k (least significant) correspond to clauses

- Theorem: SUBSET-SUM $\in$ P $\Rightarrow$ 3SAT $\in$ P
- Definition of R:
- "On input φ with v variables and k clauses :
- For each variable x include

  a_x^T = 1 in x's digit, and 1 in every digit of a clause where x appears without negation

  a_x^F = 1 in x's digit, and 1 in every digit of a clause where x appears negated
- For each clause C, include twice

  a_C = 1 in C's digit, and 0 in others
- Set t = 1 in first v digits, and 3 in rest k digits"

Example:

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

3 variables + 3 clauses $\Rightarrow$ 6 digits for each number

	var x	var y	var z	clause 1	clause 2	clause 3
a_x^T =	1	0	0	1	0	1
a_x^F =	1	0	0	0	1	0
a_y^T =	0	1	0	1	0	1
a_y^F =	0	1	0	0	1	0
a_z^T =	0	0	1	1	1	0
a_z^F =	0	0	1	0	0	1
a_{c1} =	0	0	0	1	0	0
a_{c2} =	0	0	0	0	1	0
a_{c3} =	0	0	0	0	0	1
t =	1	1	1	3	3	3

} two copies of each of these (a_{c1}, a_{c2}, a_{c3})

- Claim: $\varphi \in$ 3SAT $\Leftrightarrow$ R(φ) $\in$ SUBSET-SUM
- Proof: $\Rightarrow$

  Suppose φ has satisfying assignment
- Pick a_x^T if x is true, a_x^F if x is false
- The sum of these numbers yield 1 in first v digits

  because ???

- Claim: $\varphi \in$ 3SAT ⇔ R(φ) $\in$ SUBSET-SUM
- Proof: ⇨

  Suppose φ has satisfying assignment
- Pick a_x^T if x is true, a_x^F if x is false
- The sum of these numbers yield: 1 in first v digits

  because a_x^T , a_x^F have 1 in x's digit, 0 in others

  and 1, 2, or 3 in last k digits

  because ???

- Claim: $\varphi \in$ 3SAT ⬄ R(φ) $\in$ SUBSET-SUM
- Proof: ⇨

  Suppose φ has satisfying assignment
- Pick a_x^T if x is true, a_x^F if x is false
- The sum of these numbers yield 1 in first v digits

  because a_x^T , a_x^F have 1 in x's digit, 0 in others

  and 1, 2, or 3 in last k digits

  because each clause has true literal, and

  a_x^T has 1 in clauses where x appears not negated

  a_x^F has 1 in clauses where x appears negated
- By picking ???? ????? ??????? ?? sum reaches t

- Claim: $\varphi \in$ 3SAT $\Leftrightarrow$ R(φ) $\in$ SUBSET-SUM
- Proof: $\Rightarrow$

  Suppose φ has satisfying assignment
- Pick a_x^T if x is true, a_x^F if x is false
- The sum of these numbers yield 1 in first v digits

  because a_x^T , a_x^F have 1 in x's digit, 0 in others

  and 1, 2, or 3 in last k digits

  because each clause has true literal, and

  a_x^T has 1 in clauses where x appears not negated

  a_x^F has 1 in clauses where x appears negated
- By picking appropriate subset of a_C sum reaches t

- Claim: $\varphi \in$ 3SAT ⇔ R(φ) $\in$ SUBSET-SUM
- Proof: ⇐
- Suppose a subset sums to t = 11111111113333333333
- No carry in sum, because ???

- Claim: $\varphi \in$ 3SAT $\Leftrightarrow$ R(φ) $\in$ SUBSET-SUM
- Proof: $\Leftarrow$
- Suppose a subset sums to t = 11111111111333333333
- No carry in sum, because only 3 literals per clause
- So digits behave “independently”
- For each pair a_x^T a_x^F exactly one is included

  otherwise ???

- Claim: $\varphi \in 3SAT \Leftrightarrow R(\varphi) \in$ SUBSET-SUM
- Proof: $\Leftarrow$
- Suppose a subset sums to t = 11111111111333333333
- No carry in sum, because only 3 literals per clause
- So digits behave "independently"
- For each pair a_x^T a_x^F exactly one is included

  otherwise would not get 1 in that digit
- Define x true if a_x^T included, false otherwise
- For any clause C, the a_C contribute ≤ 2 in C's digit
- So each clause must have a true literal

  otherwise ???

- Claim: $\varphi \in$ 3SAT $\Leftrightarrow$ R(φ) $\in$ SUBSET-SUM
- Proof: $\Leftarrow$
- Suppose a subset sums to t = 11111111111333333333
- No carry in sum, because only 3 literals per clause
- So digits behave "independently"
- For each pair a_x^T a_x^F exactly one is included

  otherwise would not get 1 in that digit
- Define x true if a_x^T included, false otherwise
- For any clause C, the a_C contribute ≤ 2 in C's digit
- So each clause must have a true literal

  otherwise sum would not get 3 in that digit

Back to example:

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

	var x	var y	var z	clause 1	clause 2	clause 3
$a_x^T =$	1	0	0	1	0	1
$a_x^F =$	1	0	0	0	1	0
$a_y^T =$	0	1	0	1	0	1
$a_y^F =$	0	1	0	0	1	0
$a_z^T =$	0	0	1	1	1	0
$a_z^F =$	0	0	1	0	0	1
(2x) $a_{c1} =$	0	0	0	1	0	0
(2x) $a_{c2} =$	0	0	0	0	1	0
(2x) $a_{c3} =$	0	0	0	0	0	1
$t =$	1	1	1	3	3	3

Back to example:

$$\varphi = (x \lor y \lor z) \land (\neg x \lor \neg y \lor z) \land (x \lor y \lor \neg z)$$

0 1 0 1 0 0 0 1 1

	var x	var y	var z	clause 1	clause 2	clause 3	
a_x^T =	1	0	0	1	0	1	
a_x^F =	**1**	**0**	**0**	**0**	**1**	**0**	
a_y^T =	**0**	**1**	**0**	**1**	**0**	**1**	
a_y^F =	0	1	0	0	1	0	
a_z^T =	0	0	1	1	1	0	
a_z^F =	**0**	**0**	**1**	**0**	**0**	**1**	
(2x) **a_{c1} =**	**0**	**0**	**0**	**1**	**0**	**0**	(choose twice)
(2x) **a_{c2} =**	**0**	**0**	**0**	**0**	**1**	**0**	(choose twice)
(2x) **a_{c3} =**	**0**	**0**	**0**	**0**	**0**	**1**	
t =	1	1	1	3	3	3	

Assignment

x = 0
y = 1
z = 0

Back to example:

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

1 1 1 0 0 1 1 1 0

		var x	var y	var z	clause 1	clause 2	clause 3	
	a_x^T =	**1**	**0**	**0**	**1**	**0**	**1**	
	a_x^F =	1	0	0	0	1	0	
	a_y^T =	**0**	**1**	**0**	**1**	**0**	**1**	
	a_y^F =	0	1	0	0	1	0	
	a_z^T =	**0**	**0**	**1**	**1**	**1**	**0**	
	a_z^F =	0	0	1	0	0	1	
(2x)	a_{c1} =	0	0	0	1	0	0	
(2x)	**a_{c2} =**	**0**	**0**	**0**	**0**	**1**	**0**	(choose twice)
(2x)	**a_{c3} =**	**0**	**0**	**0**	**0**	**0**	**1**	
	t =	1	1	1	3	3	3	

Assignment

x = 1

y = 1

z = 1

- It remains to argue that ???

- It remains to argue that R runs in polynomial time

- To compute numbers a_x^T a_x^F :

  For each variable x, examine $k \leq | \varphi |$ clauses

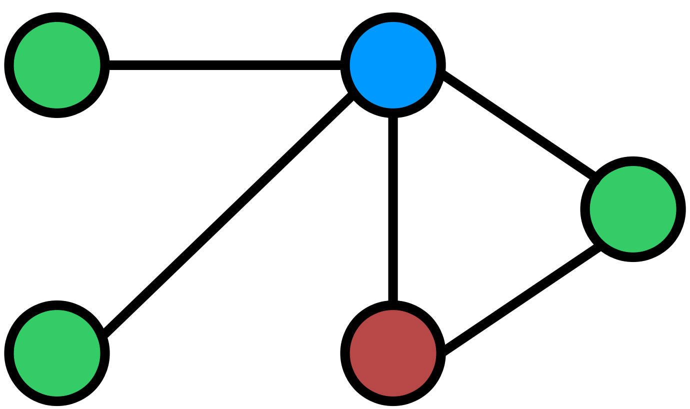
  Overall, examine $v\ k \leq | \varphi |^2$ clauses

- To compute numbers a_C examine $k \leq | \varphi |$ clauses

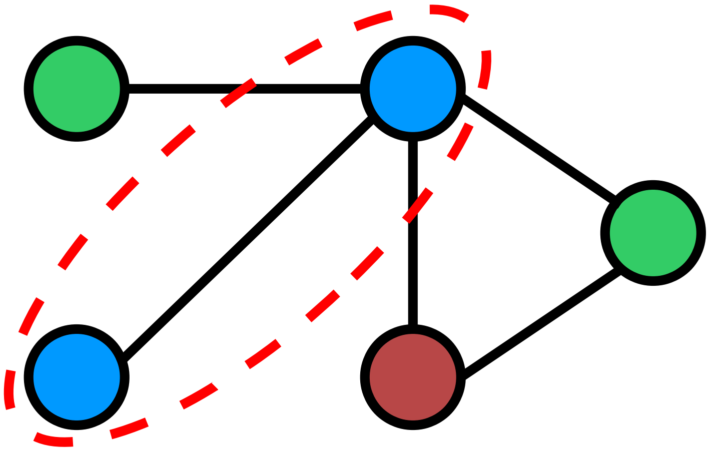
- In total $| \varphi |^2 + | \varphi |$, which is polynomial in input length
- End of proof that SUBSET-SUM $\in$ P ⇨ 3SAT $\in$ P

- Definition: A 3-coloring of a graph is a coloring of each node, using at most 3 colors, such that no adjacent nodes have the same color.

- Example:



a 3-coloring

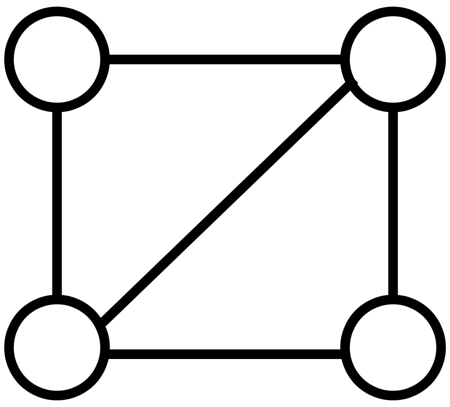


not a 3-coloring

- Definition:

  3COLOR = {G | G is a graph with a 3-coloring}

- Example:

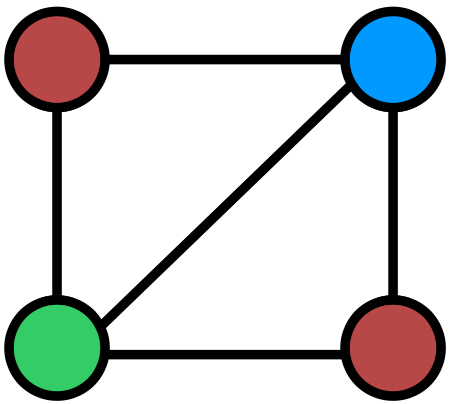
G = 

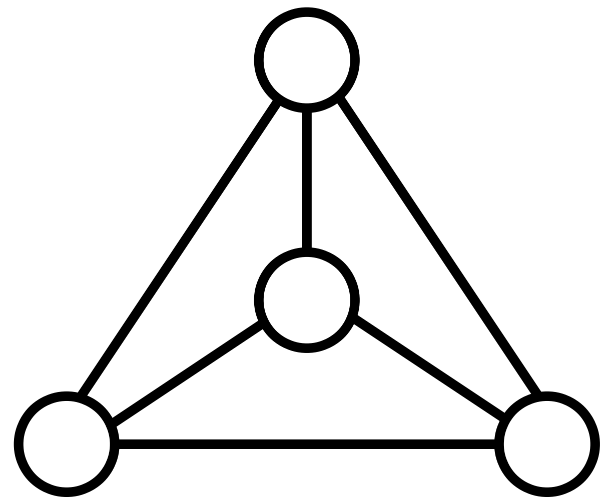
G ?? 3COLOR

- Definition:

3COLOR = {G | G is a graph with a 3-coloring}

- Example:

G = 

H = 

G $\in$ 3COLOR

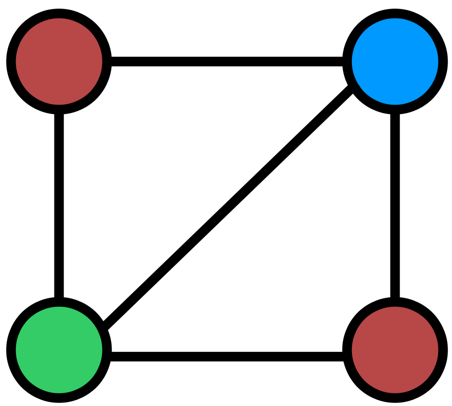
H ? 3COLOR

- Definition:

  3COLOR = {G | G is a graph with a 3-coloring}

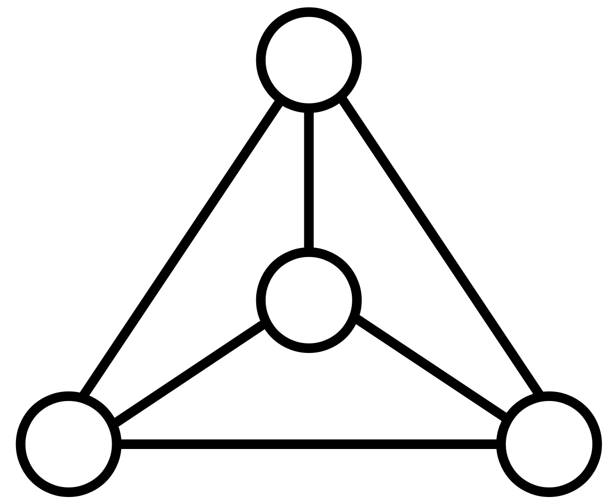
- Example:

G =



$G \in$ 3COLOR

H =



$H \notin$ 3COLOR
(> 3 nodes, all connected)

- Conjecture: 3COLOR $\notin$ P

- **Theorem**: 3COLOR $\in$ P $\Rightarrow$ 3SAT $\in$ P

- **Proof outline**:

Give algorithm R that on input φ :

(1) Computes a graph G_φ such that

$$\varphi \in 3SAT \Leftrightarrow G_\varphi \in 3COLOR.$$

(2) R runs in polynomial time

Enough to prove the theorem ?

- Theorem: 3COLOR $\in$ P $\Rightarrow$ 3SAT $\in$ P

- Proof outline:

Give algorithm R that on input φ :

(1) Computes a graph G_φ such that

$$\varphi \in 3SAT \Leftrightarrow G_\varphi \in 3COLOR.$$

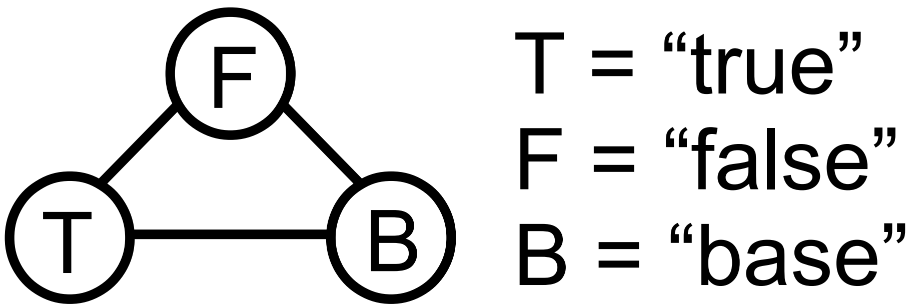
(2) R runs in polynomial time

Enough to prove the theorem because:

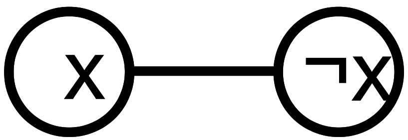
If $\exists$ TM C that solves 3COLOR in polynomial-time

Then C(R(φ)) solves 3SAT in polynomial-time

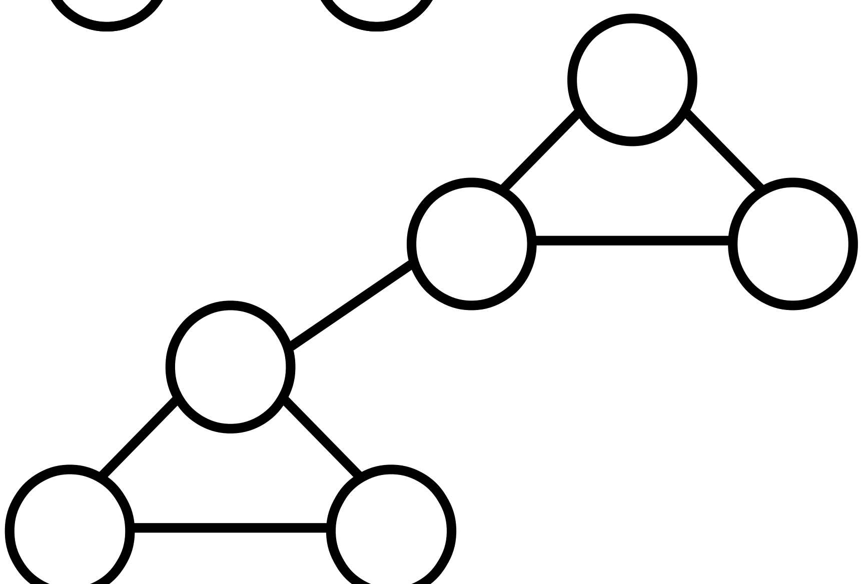
- Theorem: 3COLOR $\in$ P $\Rightarrow$ 3SAT $\in$ P
- Definition of R:
  - "On input φ, construct G_φ as follows:
  - Add 3 special nodes called the "palette".



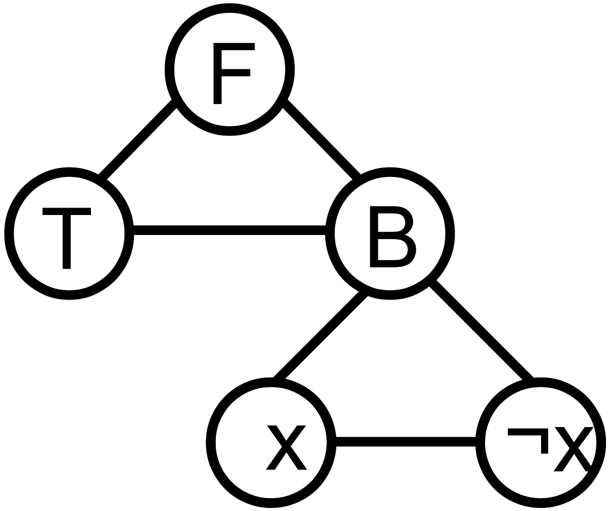
  - For each variable, add 2 literal nodes.



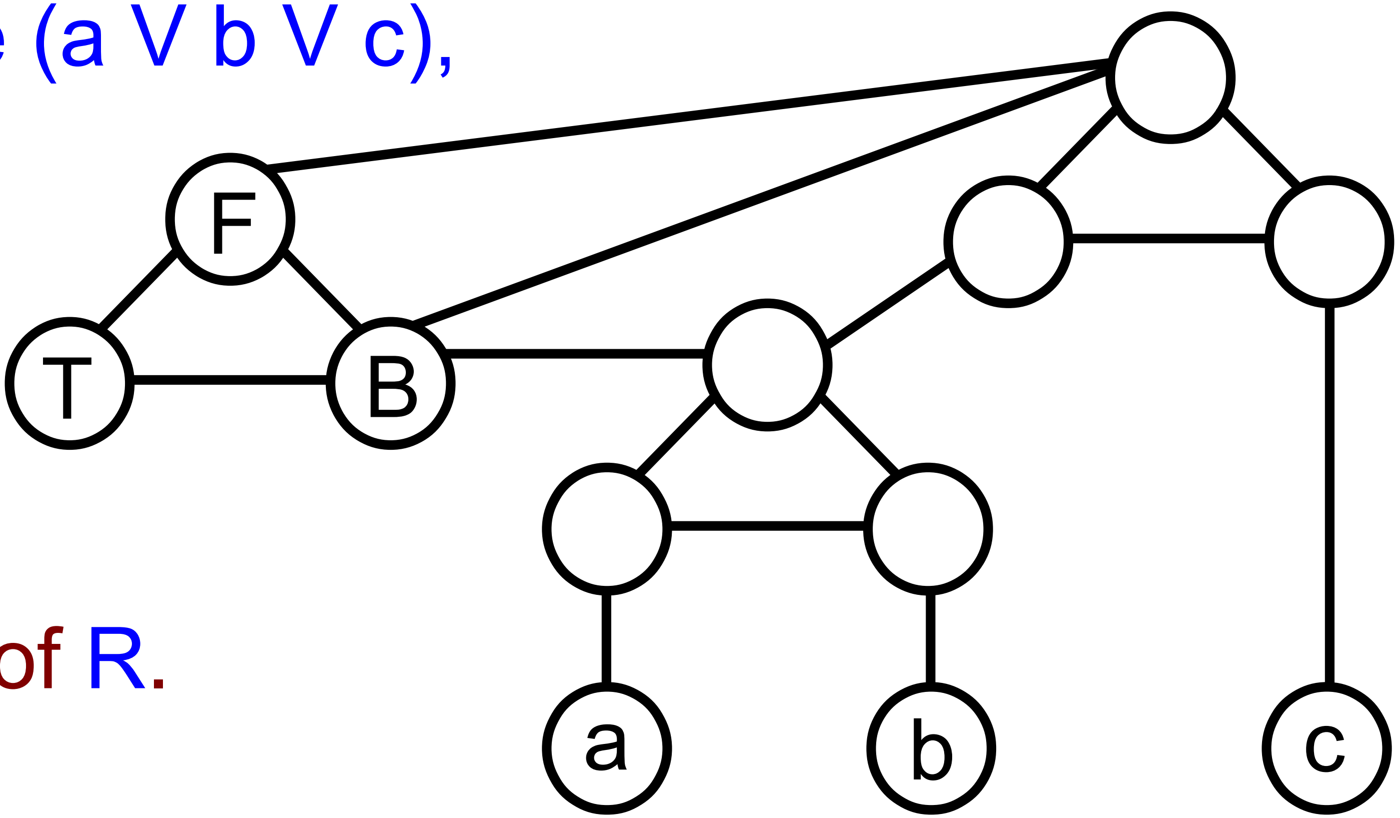
  - For each clause, add 6 clause nodes.



- **Theorem**: 3COLOR $\in$ P $\Rightarrow$ 3SAT $\in$ P
- Definition of R (continued):
  - For each variable x, connect:

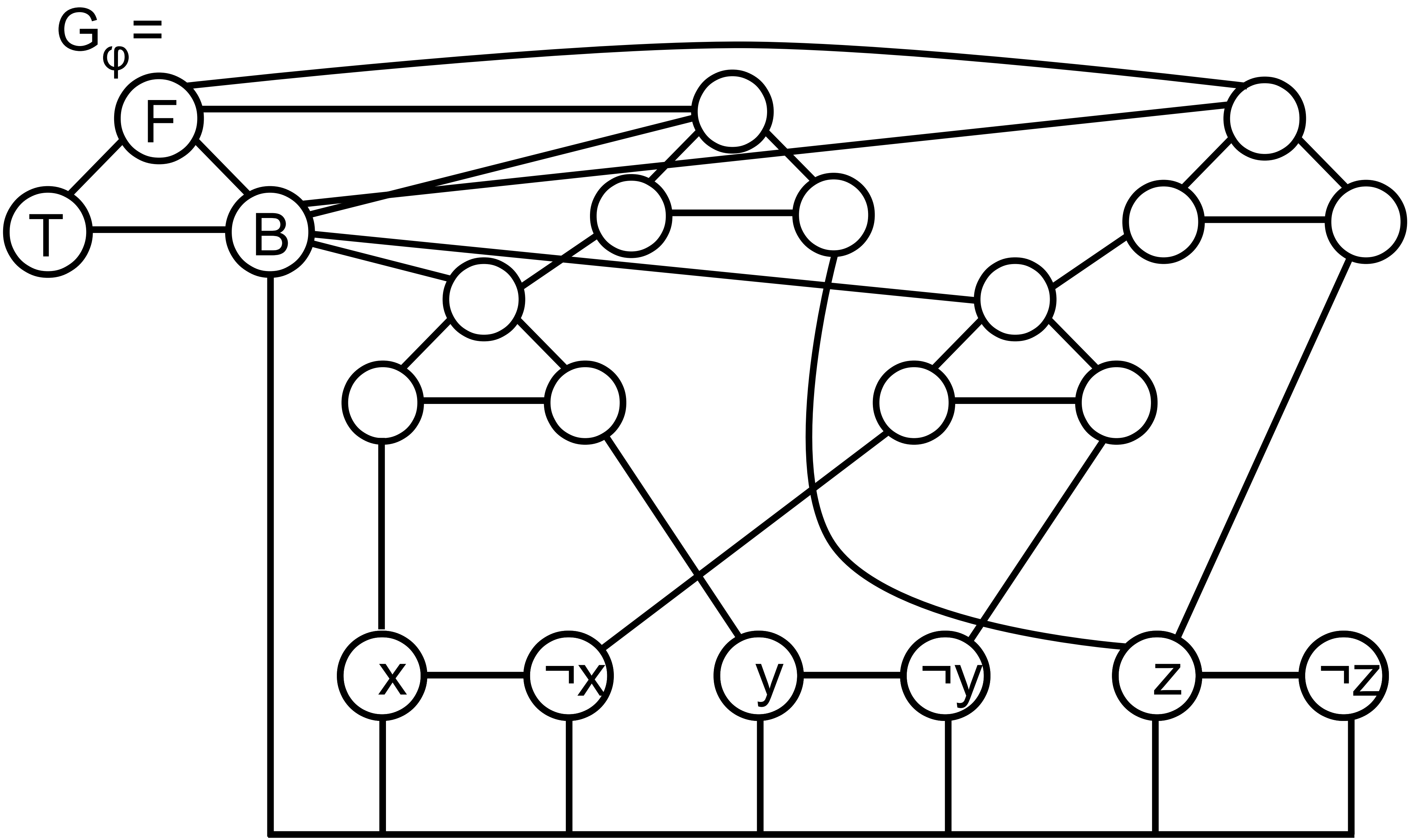


  - For each clause (a V b V c), connect:



- End of definition of R.

Example: $\varphi = (x \vee y \vee z) \wedge (\neg x \vee \neg y \vee z)$



- Claim: $\varphi \in 3SAT \Leftrightarrow G_\varphi \in 3COLOR$

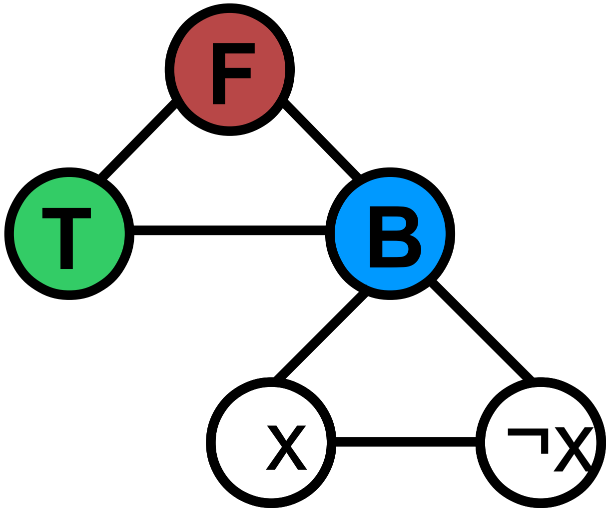
- Before proving the claim, we make some remarks,

- and prove a Fact that will be useful

Remark

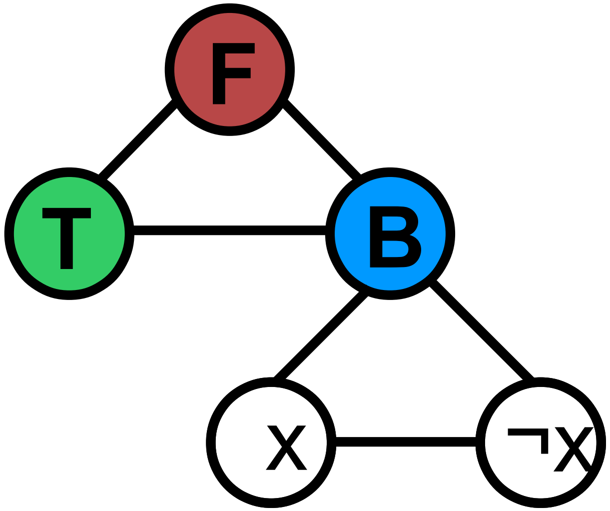
- Idea: T's color represents TRUE
  F's color represents FALSE

  - In a 3-coloring, all variable nodes must be colored T or F because?



Remark

- Idea: T's color represents TRUE
  F's color represents FALSE

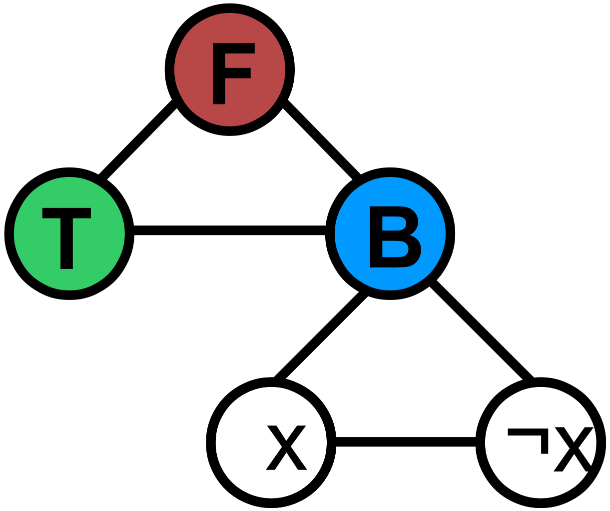


- In a 3-coloring, all variable nodes must be colored **T** or **F** because connected to **B**.

  Also, x and ¬x must have different colors because?

Remark

- Idea: T's color represents TRUE
  F's color represents FALSE

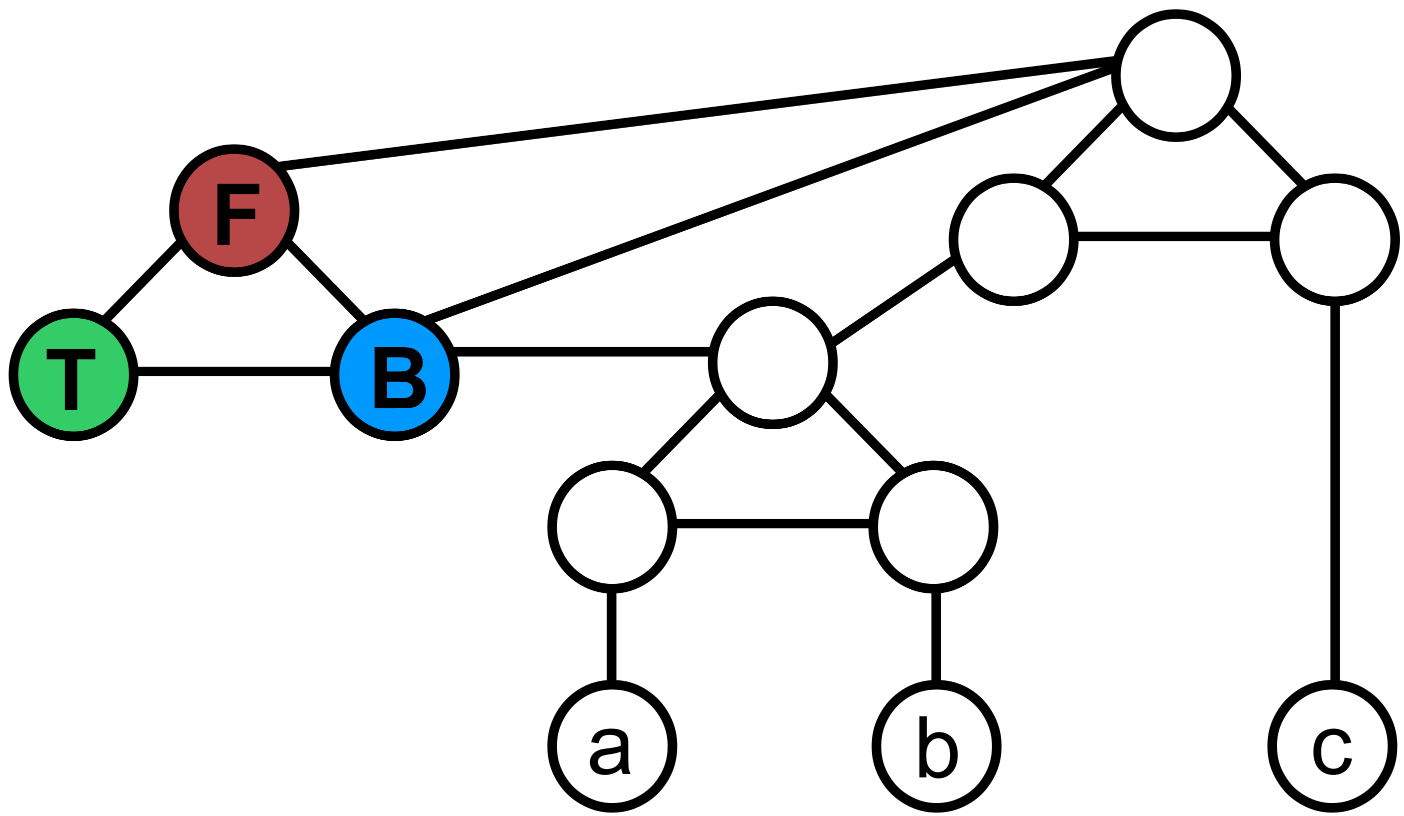


- In a 3-coloring, all variable nodes must be colored T or F because connected to B.

  Also, x and ¬x must have different colors because they are connected.

  So we can "translate" a 3-coloring of G_φ into a true/false assignment to variables of φ

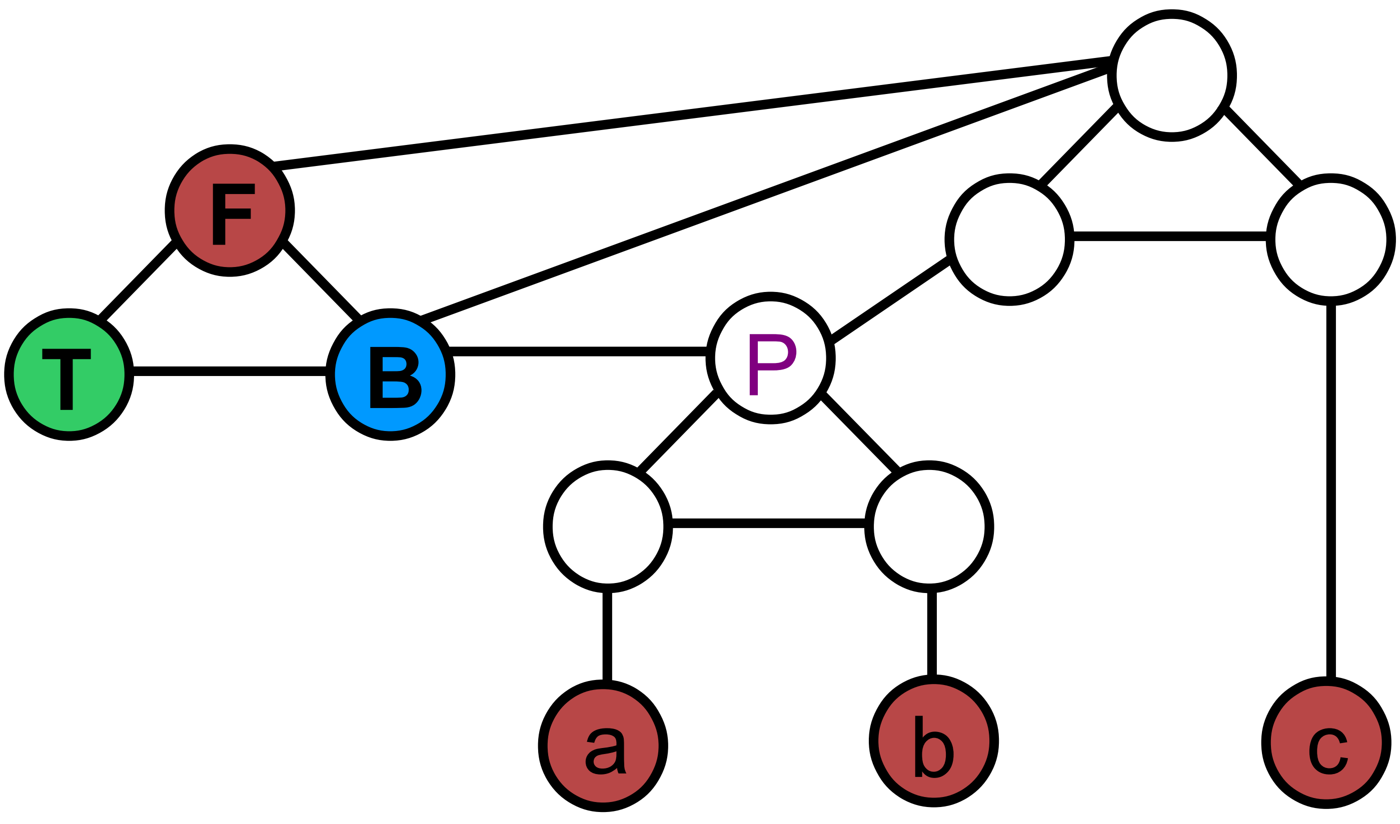
Fact: Graph below 3-colorable ⇔ a, b, or c colored T



Fact: Graph below 3-colorable ⇔ a, b, or c colored T

Proof of ⇒: Suppose by contradiction that

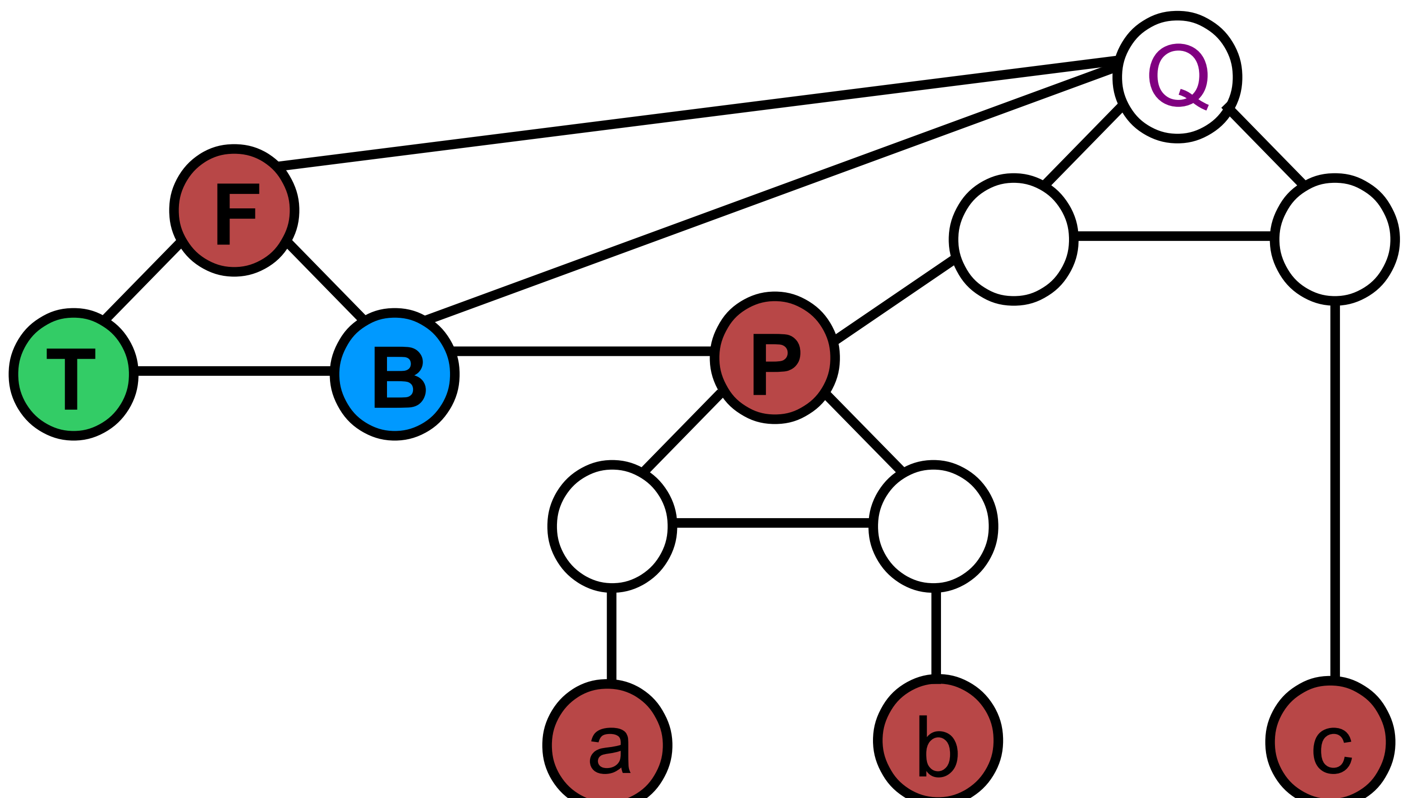
a, b, and c are all colored F then P colored how?



Fact: Graph below 3-colorable ⇔ a, b, or c colored T

Proof of ⇒: Suppose by contradiction that

a, b, and c are all colored F then P colored F.
Then Q colored how?

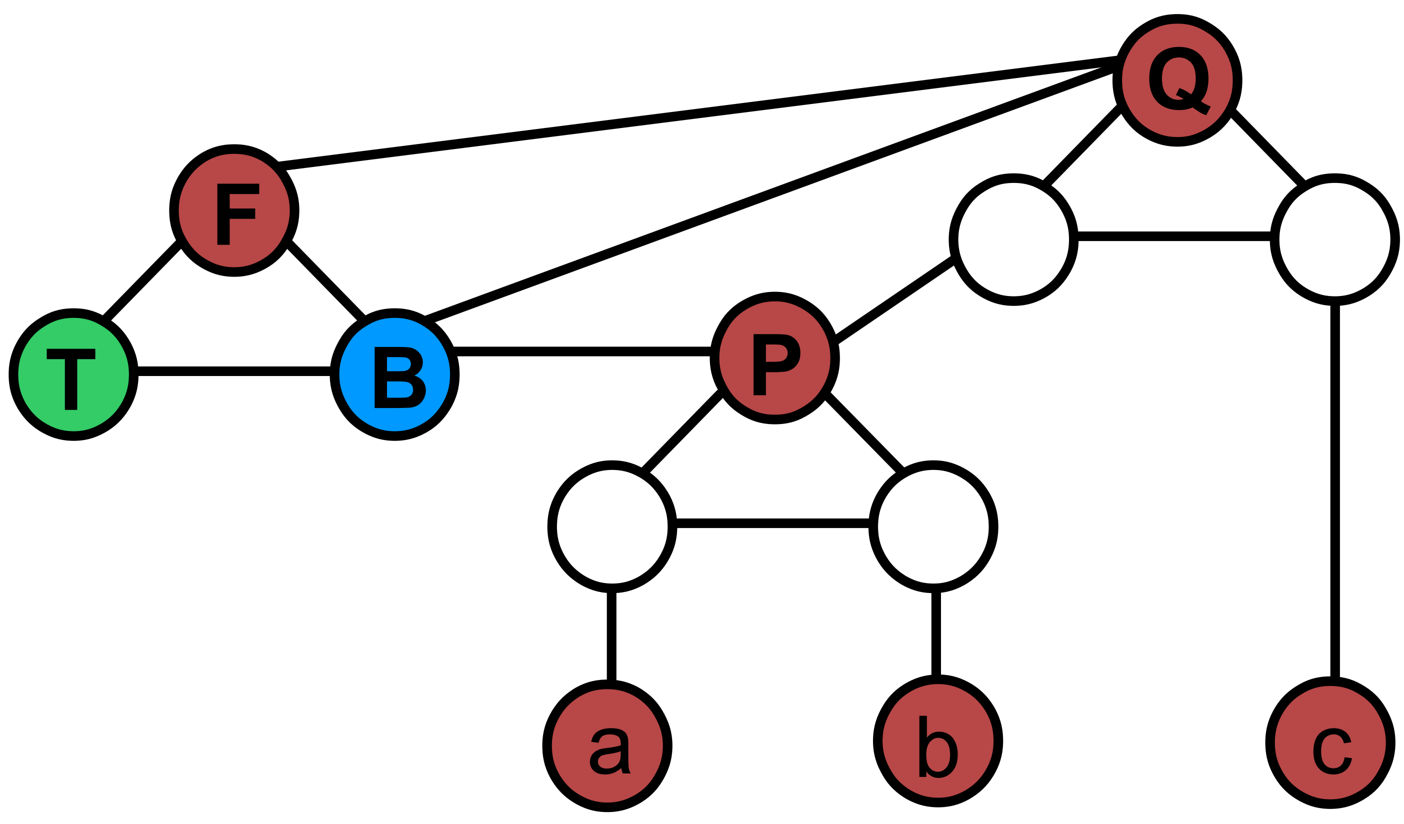


Fact: Graph below 3-colorable ⇔ a, b, or c colored T

Proof of ⇒: Suppose by contradiction that

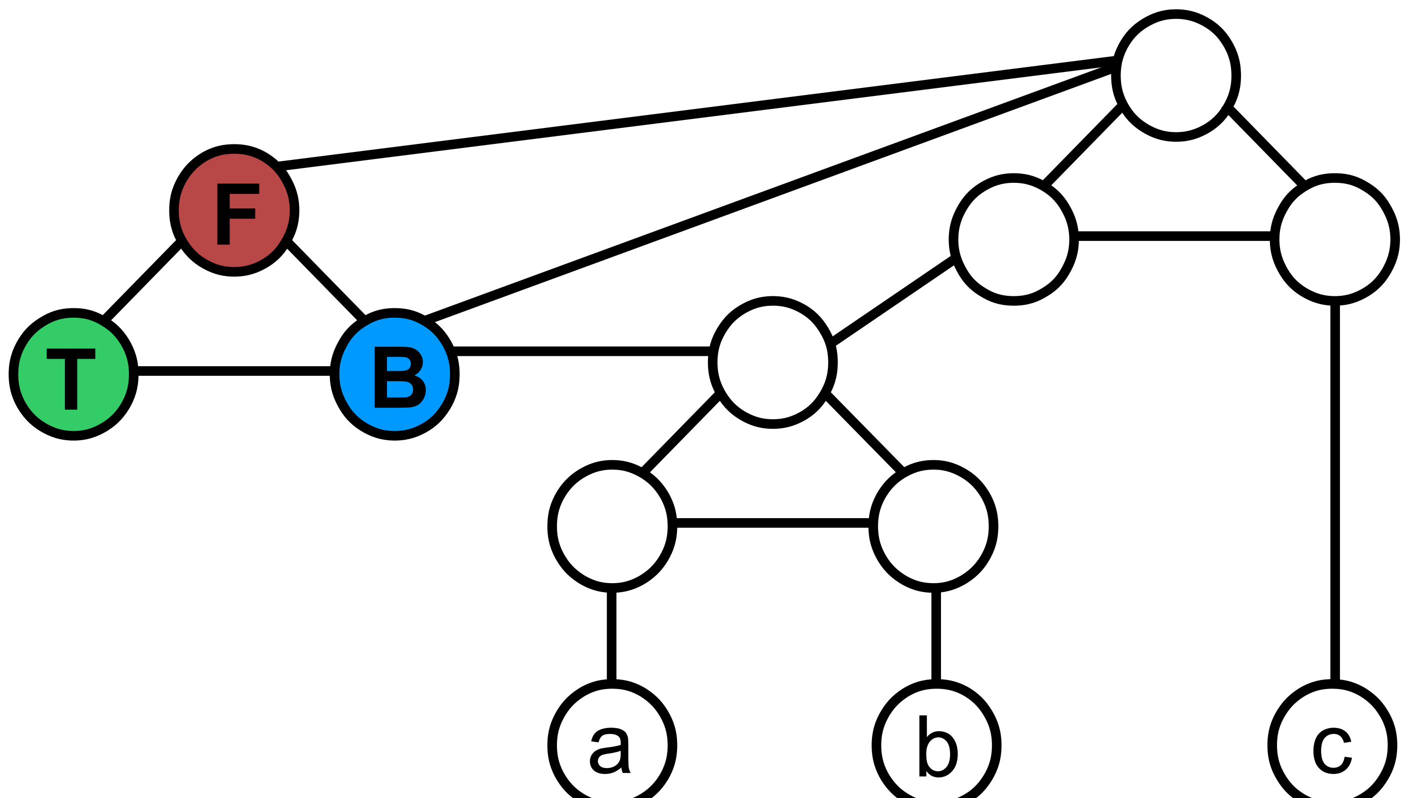
a, b, and c are all colored F then P colored F.
Then Q colored F. But this is not a valid 3-coloring

Done



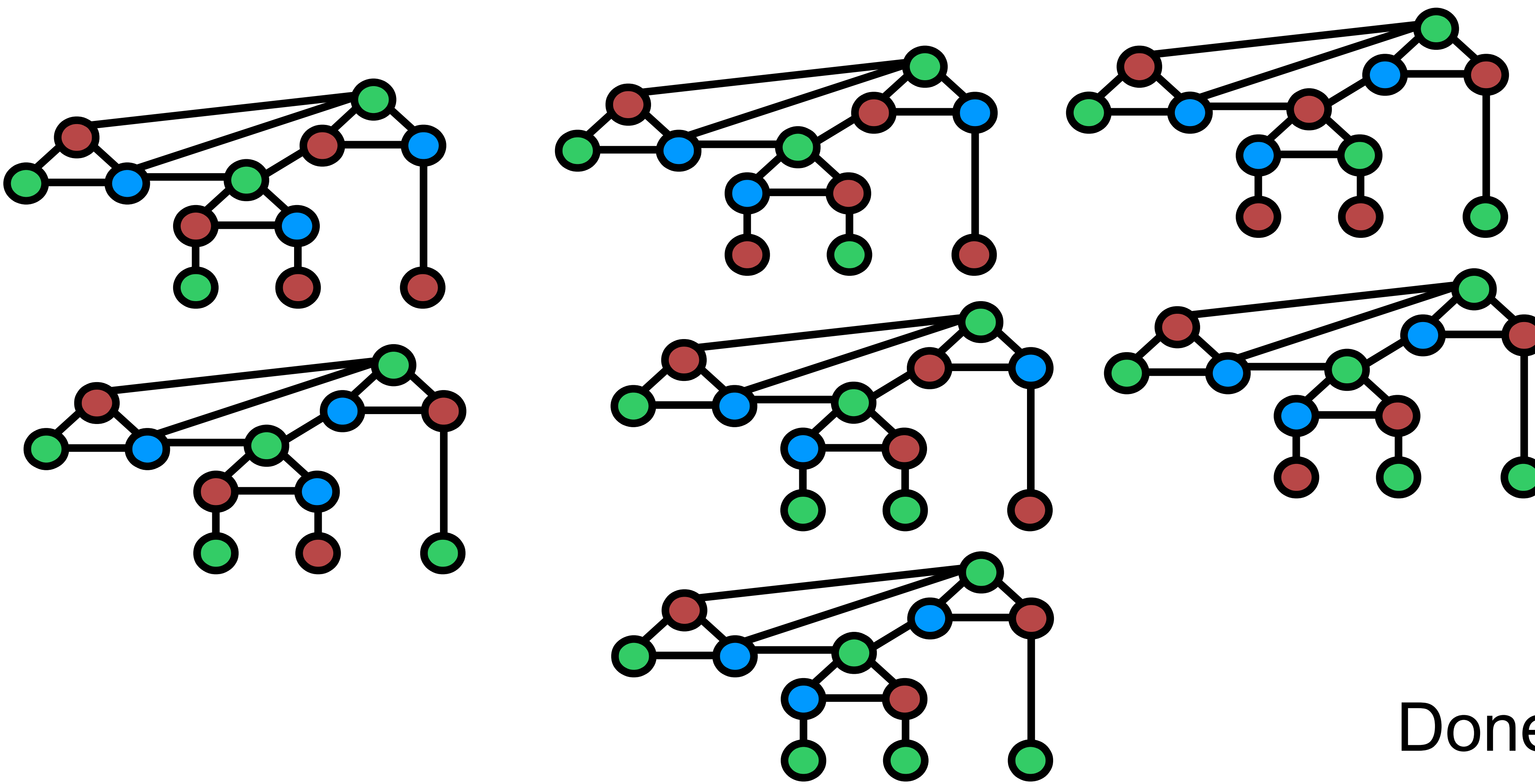
Fact: Graph below 3-colorable ⇔ a, b, or c colored T

Proof of ⇐: We show a 3-coloring for each way in which a, b, and c may be colored



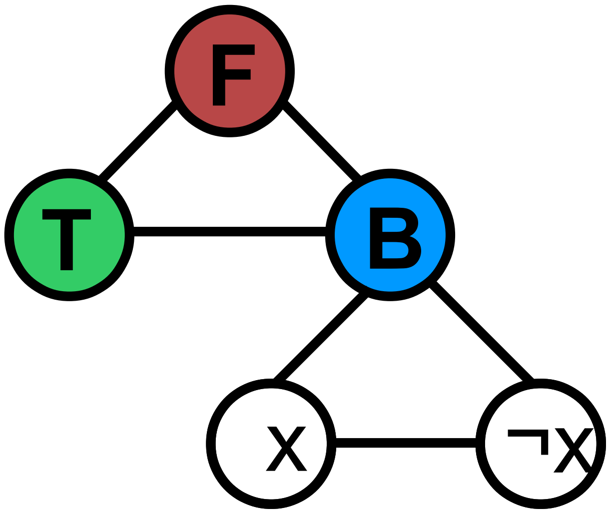
Fact: Graph below 3-colorable ⇔ a, b, or c colored T

Proof of ⇐: We show a 3-coloring for each way in which a, b, and c may be colored

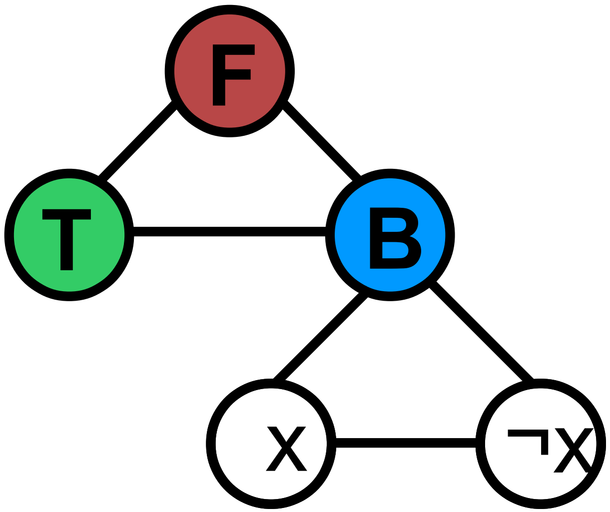


Done

- Claim: $\varphi \in$ 3SAT $\Leftrightarrow G_\varphi \in$ 3COLOR
- Proof: $\Rightarrow$
- Color palette nodes green, red, blue: T, F, B.
- Suppose φ has satisfying assignment.
- Color literal nodes T or F accordingly
  Ok because ?



- Claim: $\varphi \in 3SAT \Leftrightarrow G_\varphi \in 3COLOR$
- Proof: ⇨
- Color palette nodes green, red, blue: T, F, B.
- Suppose φ has satisfying assignment.
- Color literal nodes T or F accordingly
  Ok because they don't touch
  T or F in palette, and x and ¬ x
  are given different colors



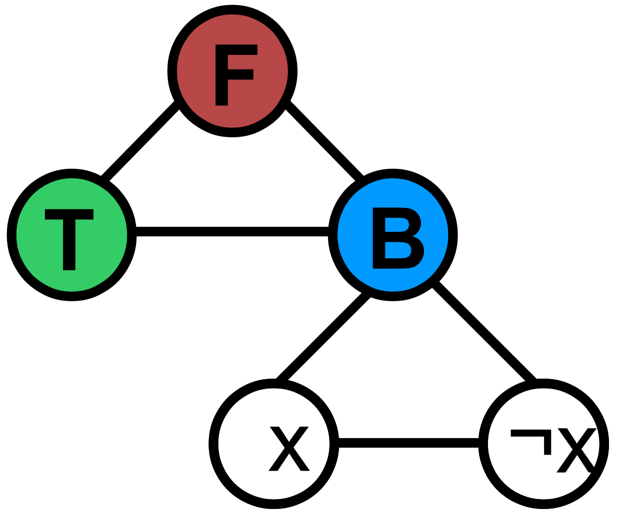
- Color clause nodes using previous Fact.
  Ok because?

- Claim: $\varphi \in$ 3SAT ⇔ $G_\varphi \in$ 3COLOR
- Proof: ⇨
- Color palette nodes green, red, blue: T, F, B.
- Suppose φ has satisfying assignment.
- Color literal nodes T or F accordingly

  Ok because they don't touch

  T or F in palette, and x and ¬ x

  are given different colors



- Color clause nodes using previous Fact.

  Ok because each clause has some true literal

- Claim: $\varphi \in 3SAT \Leftrightarrow G_\varphi \in 3COLOR$
- Proof: ⇦
- Suppose G_φ has a 3-coloring
- Assign all variables to **true** or **false** accordingly. This is a valid assignment because?

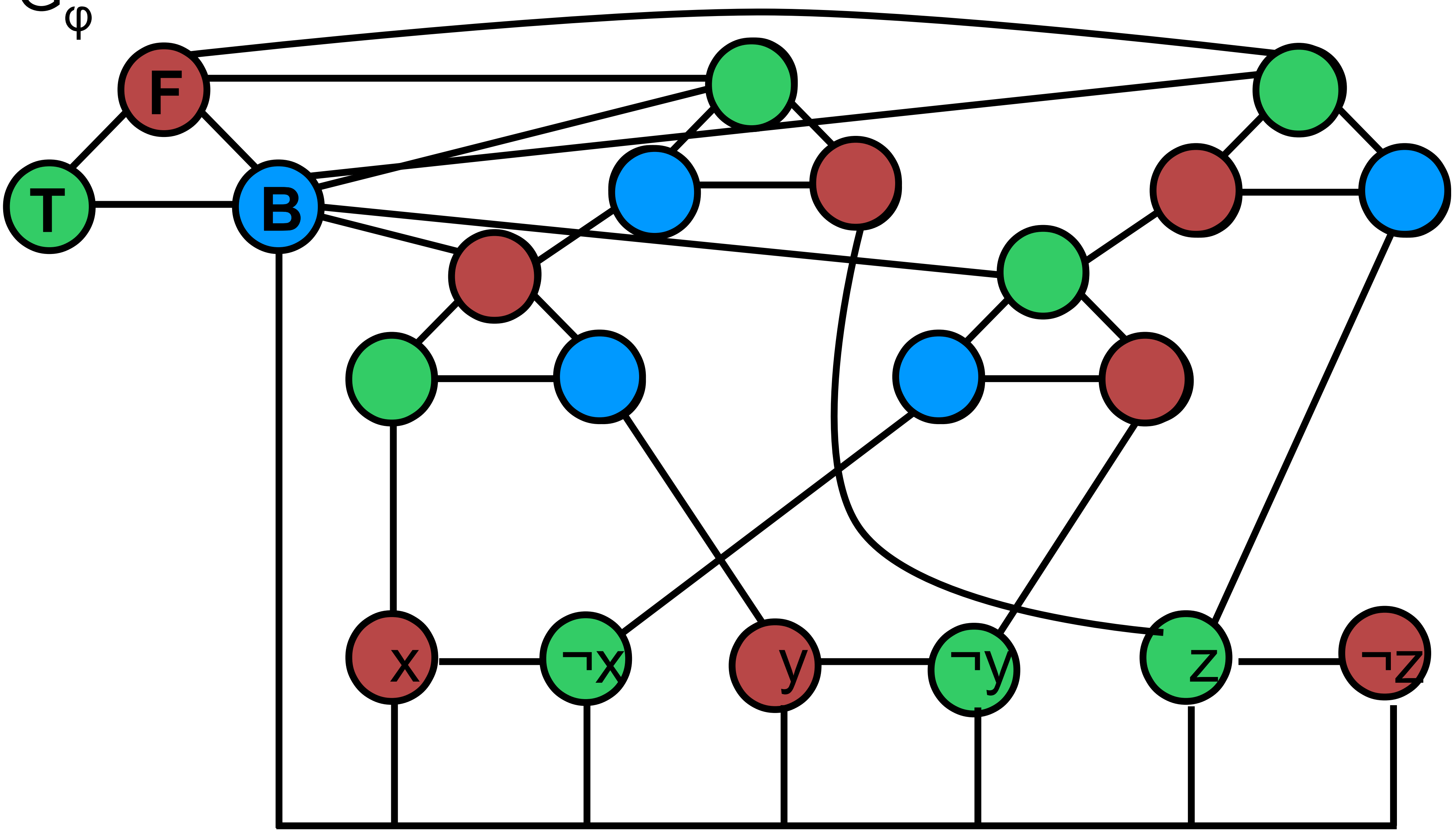
- Claim: $\varphi \in 3SAT \Leftrightarrow G_\varphi \in 3COLOR$
- Proof: ⇦
- Suppose G_φ has a 3-coloring
- Assign all variables to **true** or **false** accordingly. This is a valid assignment because by Remark, x and ¬x are colored **T** or **F** and don't conflict.
- This gives some true literal per clause because?

- Claim: $\varphi \in$ 3SAT ⇔ $G_\varphi \in$ 3COLOR
- Proof: ⇐
- Suppose G_φ has a 3-coloring
- Assign all variables to **true** or **false** accordingly. This is a valid assignment because by Remark, x and ¬x are colored T or F and don't conflict.
- This gives some true literal per clause because clause is colored correctly, and by previous Fact
- All clauses are satisfied, so φ is satisfied.

Example: $\varphi = (x \lor y \lor z) \land (\neg x \lor \neg y \lor z)$

Satisfying assignment: x = 0, y = 0, z = 1

$G_\varphi =$



- It remains to argue that ???

- It remains to argue that R runs in polynomial time

- To add variable nodes and edges, cycle over $v \leq | \varphi |$ variables

- To add clause nodes and edges, cycle over $c \leq | \varphi |$ clauses

- Overall, $\leq | \varphi | + | \varphi |$, which is polynomial in input length $| \varphi |$

- This is the only interesting detail
- Conclude proof that 3COLOR $\in$ P ⇨ 3SAT $\in$ P

- We saw polynomial-time reductions
  from 3SAT to CLIQUE
  SUBSET-SUM
  3COLOR

- There are many other polynomial-time reductions

- They form a fascinating web

- Coming up with reductions is “art”

Big picture

- All languages
- Decidable

  Turing machines
- NP
- P
- Context-free

  Context-free grammars, push-down automata
- Regular

  Automata, non-deterministic automata, regular expressions

- Definition: NP =

  { L : ∃ integer c, ∃ TM M that runs in time n^c :

  $w \in L \Leftrightarrow \exists y , |y| \le |w|^c$, M accepts (w,y) }

- y is called "witness"

- NP means Non-deterministic Polynomial time.

  "Non-deterministic" refers to "∃ y"

- Do not confuse NP with (not P)

- Definition: NP =

{ L : $\exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y$, $|y| \le |w|^c$, M accepts (w,y) }

- Claim: P $\subseteq$ NP
- Proof:

?

- Definition: NP =

{ L : $\exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y$, $|y| \le |w|^c$, M accepts (w,y) }

- Claim: P $\subseteq$ NP
- Proof:

Ignore y Done

- Let us see again why P $\subseteq$ NP

- NP = { L : $\exists$ integer c, $\exists$ TM M that runs in time n^c :

  w $\in$ L $\Leftrightarrow$ $\exists$ y , $|y| \leq |w|^c$, M accepts (w,y) }

- P := ?

- Let us see again why P $\subseteq$ NP

- NP = { L : $\exists$ integer c, $\exists$ TM M that runs in time n^c :

  w $\in$ L $\Leftrightarrow$ $\exists$ y , $|y| \le |w|^c$, M accepts (w,y) }

- P := $\cup_c$ TIME(n^c) = TIME(n^1) $\cup$ TIME(n^2) $\cup$ …

  =

- Let us see again why $P \subseteq NP$

- NP = { L : $\exists$ integer c, $\exists$ TM M that runs in time n^c :

  $w \in L \Leftrightarrow \exists y$, $|y| \leq |w|^c$, M accepts (w,y) }

- P := U_c TIME(n^c) = TIME(n^1) U TIME(n^2) U …

  = {L : $\exists$ integer c : $L \in$ TIME (n^c) }

  =

- Let us see again why $P \subseteq NP$

- NP = { L : $\exists$ integer c, $\exists$ TM M that runs in time n^c :

  $w \in L \Leftrightarrow \exists y$, $|y| \le |w|^c$, M accepts (w,y) }

- P := U_c TIME(n^c) = TIME(n^1) U TIME(n^2) U …

  = {L : $\exists$ integer c : L $\in$ TIME (n^c) }

  = {L : $\exists$ integer c, $\exists$ TM M that runs in time n^c :

  M decides L }

- Let us see again why P $\subseteq$ NP

- NP = { L : $\exists$ integer c, $\exists$ TM M that runs in time n^c :

  w $\in$ L $\Leftrightarrow$ $\exists$ y , $|y| \leq |w|^c$, M accepts (w,y) }

- P := $\cup_c$ TIME(n^c) = TIME(n^1) U TIME(n^2) U …

  = {L : $\exists$ integer c : L $\in$ TIME (n^c) }

  = {L : $\exists$ integer c, $\exists$ TM M that runs in time n^c :

  w $\in$ L <=> M accepts w}

- Same definition, except for " $\exists$ y " part

- Definition: NP =

$\{ L : \exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y$, $|y| \leq |w|^c$, M accepts (w,y) $\}$

- Claim: 3SAT $\in$ NP
- Proof: Input $w = \varphi$. y is ?

- Definition: NP =

$\{ L : \exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y , |y| \leq |w|^c$, M accepts (w,y) $\}$

- Claim: 3SAT $\in$ NP
- Proof: Input w = φ. y is a truth assignment
- $|y| \leq$?

- Definition: NP =

$\{ L : \exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y , |y| \leq |w|^c$, M accepts (w,y) $\}$

- Claim: 3SAT $\in$ NP
- Proof: Input w = φ. y is a truth assignment
- $|y| \leq$ number of variables $\leq | \varphi |$
- M checks ?

- Definition: NP =

{ L : $\exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y$, $|y| \le |w|^c$, M accepts (w,y) }

- Claim: 3SAT $\in$ NP
- Proof: Input w = φ. y is a truth assignment
- $|y| \le$ number of variables $\le |\varphi|$
- M checks if all clauses in φ satisfied by y
- M examines ≤ ? clauses $\Rightarrow$ polynomial time

- Definition: NP =

$\{ L : \exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y, |y| \le |w|^c$, M accepts $(w,y) \}$

- Claim: 3SAT $\in$ NP
- Proof: Input w = φ. y is a truth assignment
- $|y| \le$ number of variables $\le | \varphi |$
- M checks if all clauses in φ satisfied by y
- M examines $\le | \varphi |$ clauses ⇨ polynomial time   Done

- Definition: NP =

{ L : $\exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y$, $|y| \le |w|^c$, M accepts (w,y) }

- Claim: CLIQUE $\in$ NP
- Proof: Input w = (G,t). y is ?

- Definition: NP =

{ L : $\exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y$, $|y| \leq |w|^c$, M accepts (w,y) }

- Claim: CLIQUE $\in$ NP
- Proof: Input w = (G,t). y is a set of t nodes
- $|y| \leq$?

- Definition: NP =

{ L : ∃ integer c, ∃ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y$, $|y| \le |w|^c$, M accepts (w,y) }

- Claim: CLIQUE $\in$ NP
- Proof: Input w = (G,t). y is a set of t nodes
- $|y| \le t \le |w|$
- M checks if ?

- Definition: NP =

{ L : ∃ integer c, ∃ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y, |y| \leq |w|^c$, M accepts (w,y) }

- Claim: CLIQUE $\in$ NP
- Proof: Input w = (G,t). y is a set of t nodes
- $|y| \leq t \leq |w|$
- M checks if every pair of nodes in y is connected
- M examines ≤ ? pairs ⇨ polynomial time

- Definition: NP =

$\{ L : \exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y, |y| \leq |w|^c$, M accepts (w,y) $\}$

- Claim: CLIQUE $\in$ NP
- Proof: Input w = (G,t). y is a set of t nodes
- $|y| \leq t \leq |w|$
- M checks if every pair of nodes in y is connected
- M examines $\leq t^2$ pairs $\Rightarrow$ polynomial time   Done

- Definition: NP =

{ L : ∃ integer c, ∃ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y$, $|y| \le |w|^c$, M accepts (w,y) }

- Claim: SUBSET-SUM ∈ NP
- Proof: w = $(a_1, a_2, \ldots, a_n, t)$; y is ?

- Definition: NP =

$\{ L : \exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y , |y| \leq |w|^c$, M accepts (w,y) $\}$

- Claim: SUBSET-SUM $\in$ NP
- Proof: $w = (a_1, a_2, ..., a_n, t)$; y is a subset of the a_i
- $|y| \leq$?

- Definition: NP =

$\{ L : \exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y$, $|y| \le |w|^c$, M accepts (w,y) $\}$

- Claim: SUBSET-SUM $\in$ NP
- Proof: $w = (a_1, a_2, ..., a_n, t)$; y is a subset of the a_i
- $|y| \le n \le |w|$
- M checks if ?

- Definition: NP =

$\{ L : \exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y$, $|y| \le |w|^c$, M accepts (w,y) $\}$

- Claim: SUBSET-SUM $\in$ NP
- Proof: $w = (a_1, a_2, ..., a_n, t)$; y is a subset of the a_i
- $|y| \le n \le |w|$
- M checks if y sums to t
- M sums y $\le$? numbers $\Rightarrow$ polynomial time

- Definition: NP =

$\{ L : \exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y , |y| \leq |w|^c$, M accepts $(w,y) \}$

- Claim: SUBSET-SUM $\in$ NP
- Proof: $w = (a_1, a_2, ..., a_n, t)$; y is a subset of the a_i
- $|y| \leq n \leq |w|$
- M checks if y sums to t
- M sums $y \leq |w|$ numbers $\Rightarrow$ polynomial time   Done

- Definition: NP =

$\{ L : \exists$ integer c, $\exists$ TM M that runs in time n^c :

$w \in L \Leftrightarrow \exists y$, $|y| \leq |w|^c$, M accepts (w,y) $\}$

- Claim: 3COLOR $\in$ NP
- Proof: Input w = G. y is a coloring
- $|y| \leq |G| \leq |\ w\ |$
- M checks if adjacent nodes in G have different color
- M examines $\leq |G|^2$ pairs $\Rightarrow$ polynomial time Done

- Cook-Levin Theorem: 3SAT $\in$ P $\Rightarrow$ P = NP

- Meaning, if 3SAT $\in$ P, then arbitrary NP computation can be done efficiently

- Surprising: from one problem to arbitrary computation

- Unsurprising?: Computers made of $\vee$, $\wedge$, $\neg$ gates

  That's what 3SAT is

- Definition: L is NP-complete if

  (1) $L \in NP$, and

  (2) $L \in P \Rightarrow P = NP$

- Claim: 3SAT is NP-complete
- Proof:

(1) We saw earlier $3SAT \in NP$

(2) is Cook-Levin Theorem Done

- Definition: L is NP-complete if

    (1) $L \in NP$, and

    (2) $L \in P \Rightarrow P = NP$

- Fact: Suppose L is such that:

    (1) $L \in NP$

    (2') 3SAT is polynomial-time reducible to L

  then L is NP-complete

- Proof of (2):

  $L \in P \Rightarrow$?

- Definition: L is NP-complete if

  (1) $L \in NP$, and

  (2) $L \in P \Rightarrow P = NP$

- Fact: Suppose L is such that:

  (1) $L \in NP$

  (2') 3SAT is polynomial-time reducible to L

  then L is NP-complete

- Proof of (2):

  $L \in P \Rightarrow 3SAT \in P \Rightarrow$?

  (2')

- Definition: L is NP-complete if

  (1) $L \in NP$, and

  (2) $L \in P \Rightarrow P = NP$

- Fact: Suppose L is such that:

  (1) $L \in NP$

  (2') 3SAT is polynomial-time reducible to L

  then L is NP-complete

- Proof of (2):

  $L \in P \Rightarrow 3SAT \in P \Rightarrow P = NP$ Done

  (2') (Cook-Levin Theorem)

- Fact: Suppose L is such that:

  (1) $L \in NP$

  (2') 3SAT is polynomial-time reducible to L

  then L is NP-complete

- Claim:

  CLIQUE, SUBSET-SUM, 3COLOR are NP-complete

- Proof of claim:

  We showed (1) and (2') for each of these Done

- Recap:
- If L is NP-complete then L $\in$ P $\Rightarrow$ P = NP, equivalently, P $\neq$ NP $\Rightarrow$ L $\notin$ P

- 3SAT, CLIQUE, SUBSET-SUM, 3COLOR are NP-complete

- They are the “hardest problems” in NP: If there is anything in NP that is not in P, then 3SAT, CLIQUE, SUBSET-SUM, 3COLOR $\notin$ P

- What else is NP-complete?

- Many other problems people care about

- This includes many puzzles/games

- We now list a few

- Technical remark: need to generalize puzzles/games to boards/levels of arbitrary size. Not a problem.

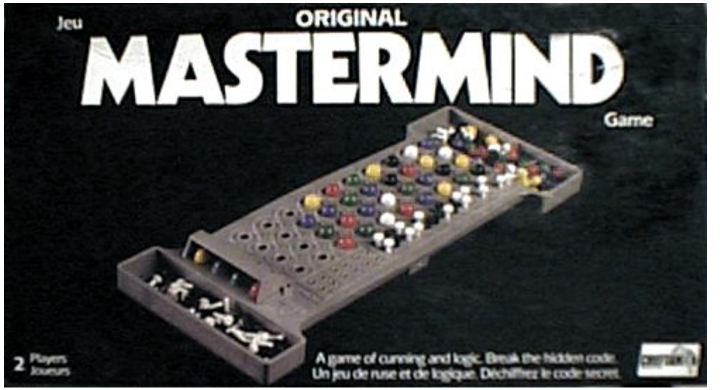
- NP-complete

- SUDOKU

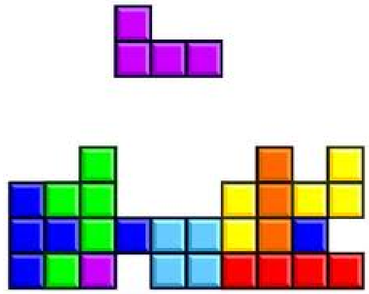
8			4		6			7
						4		
	1					6	5	
5		9		3		7	8	
				7				
	4	8		2		1		3
	5	2					9	
		1						
3			9		2			5

- PEG SOLITAIRE

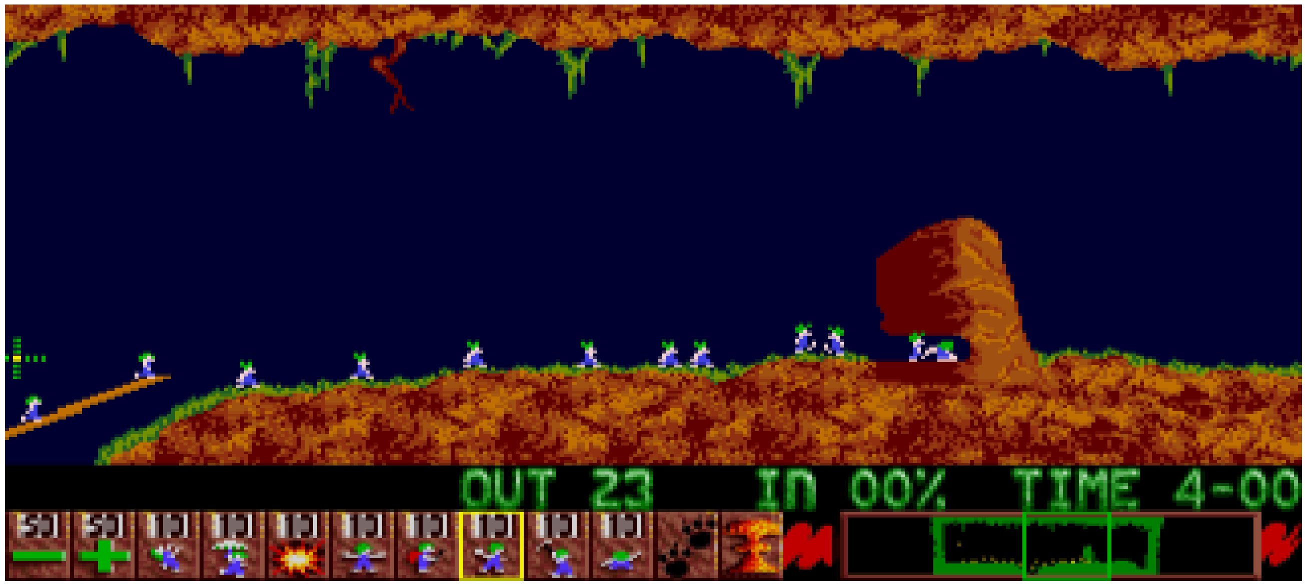
- MASTERMIND



- NP-complete



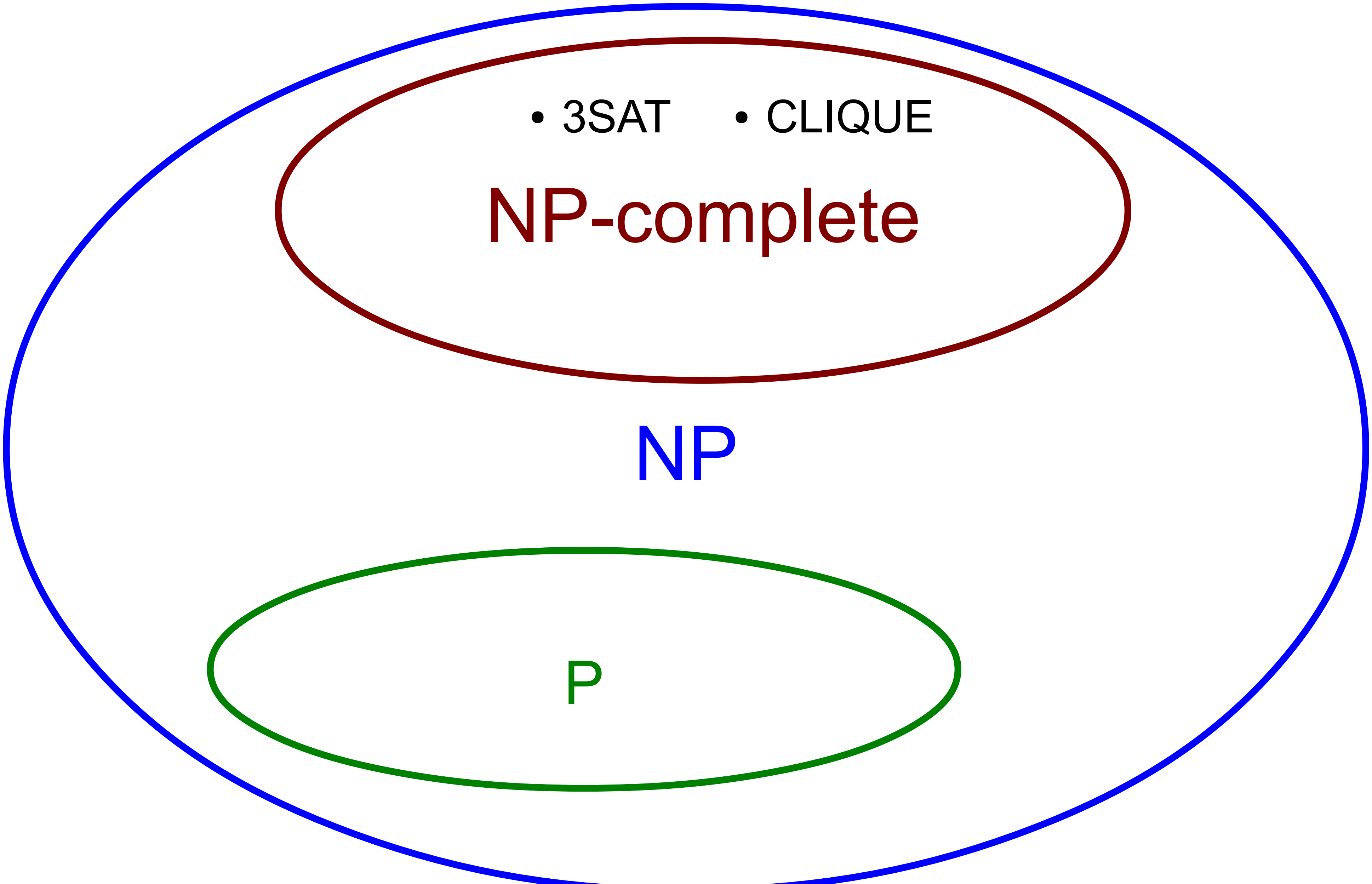
- TETRIS



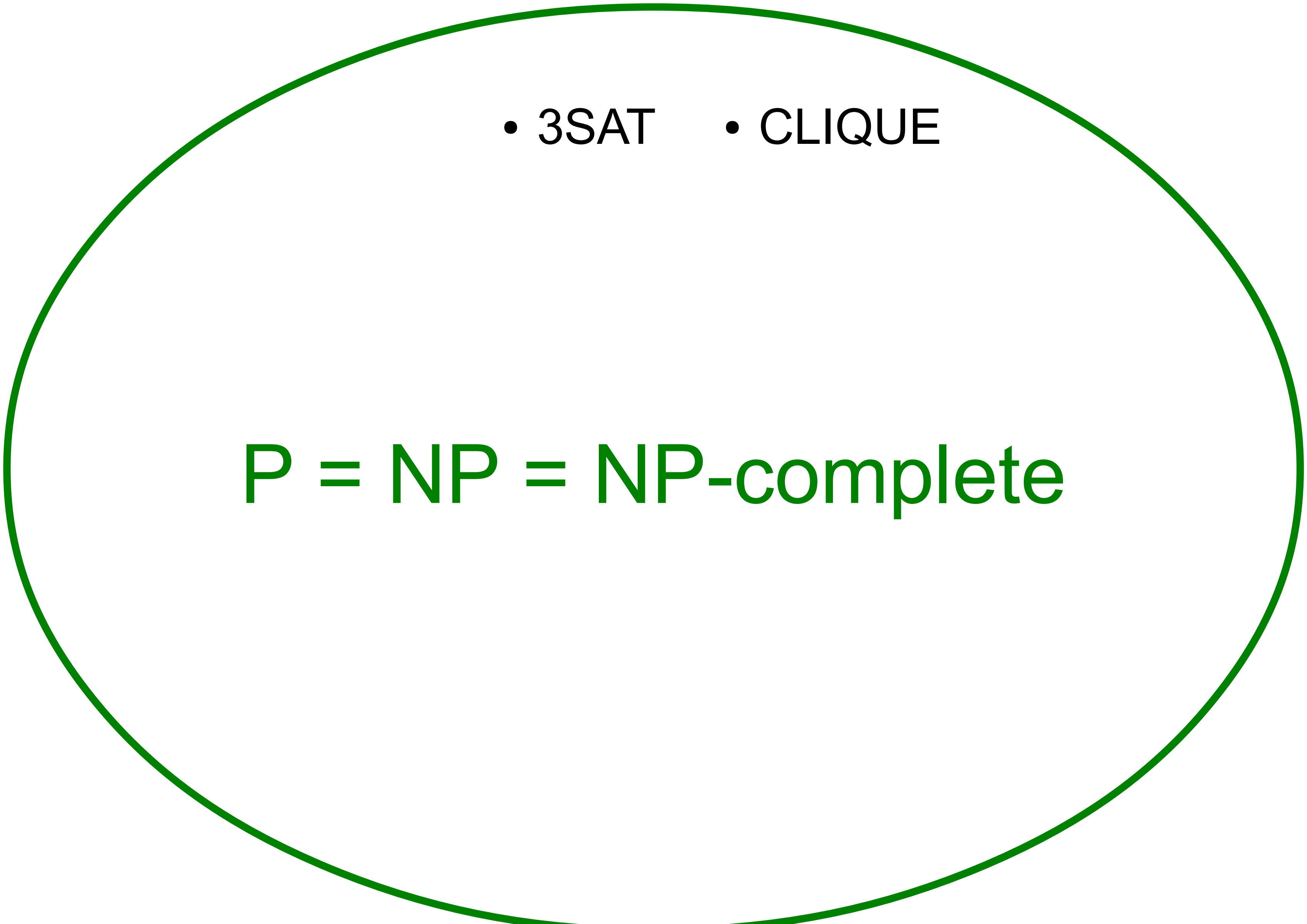
- LEMMINGS

- SUPER MARIO

Our world, assuming P ≠ NP



Our world, assuming P = NP



- Definition: Exponential Time: $\text{EXP} := \cup_c \text{TIME}(2^{n^c})$

- Claim: ? $\subseteq$ EXP

- Definition: Exponential Time: $\text{EXP} := \bigcup_c \text{TIME}(2^{n^c})$
- Recall NP = { L : $\exists$c, $\exists$TM M that runs in time n^c :

  $w \in L \Leftrightarrow \exists y$, $|y| \leq |w|^c$, M accepts (w,y) }

- Claim: NP $\subseteq$ EXP
- Proof: ?

- Definition: Exponential Time: $EXP := \cup_c TIME(2^{n^c})$
- Recall NP = { L : ∃c, ∃TM M that runs in time n^c :

  $w \in L \Leftrightarrow \exists y, |y| \le |w|^c$, M accepts (w,y) }

- Claim: $NP \subseteq EXP$
- Proof: Suppose $L \in NP$. Let c, M be as in defin. of NP

  Let TM M' := "On input w,

  for every y : $|y| \le |w|^c$, run M(w,y)

  if any accept, ACCEPT; if not, REJECT"

- M' accepts w ⇔ ?

- Definition: Exponential Time: $EXP := \cup_c TIME(2^{n^c})$
- Recall NP = { L : ∃c, ∃TM M that runs in time n^c :

  $w \in L \Leftrightarrow \exists y, |y| \le |w|^c$, M accepts (w,y) }

- Claim: $NP \subseteq EXP$
- Proof: Suppose $L \in NP$. Let c, M be as in defin. of NP.

  Let TM M' := "On input w,

  for every y : $|y| \le |w|^c$, run M(w,y)

  if any accept, ACCEPT; if not, REJECT"

- M' accepts w $\Leftrightarrow \exists y, |y| \le |w|^c$, M accepts (w,y)
- M' runs in time ?

- Definition: Exponential Time: $\text{EXP} := \cup_c \text{TIME}(2^{n^c})$
- Recall $\text{NP} = \{ L : \exists c, \exists \text{TM } M$ that runs in time n^c :

  $w \in L \Leftrightarrow \exists y , |y| \le |w|^c$, M accepts $(w,y) \}$

- Claim: $\text{NP} \subseteq \text{EXP}$
- Proof: Suppose $L \in \text{NP}$. Let c, M be as in defin. of NP.

  Let TM M' := "On input w,

  for every y : $|y| \le |w|^c$, run $M(w,y)$

  if any accept, ACCEPT; if not, REJECT"

- M' accepts $w \Leftrightarrow \exists y , |y| \le |w|^c$, M accepts (w,y)
- M' runs in time $2^{|w|^c} |(w,y)|^c \le 2^{|w|^{c+1}}$ Done

All languages

$\cup|$ Different?

Decidable

$\cup|$

EXP

$\cup|$

NP

$\cup|$

P

$\cup|$

context-free

$\cup|$

regular

All languages

U|　　　　ATM $\notin$ Decidable

Decidable

U|

EXP

U|

NP

U|

P

U|　　　　Different?

context-free

U|

regular

All languages

U| ATM $\notin$ Decidable

Decidable

U|

EXP

U|

NP

U|

P

U| $\{a^m b^m c^m : m \geq 0\} \in$ P, $\notin$ context-free

context-free

U| Different?

regular

All languages

U| ATM $\notin$ Decidable

Decidable

U| Also different (will not see)

EXP

U| Different?

NP

U| Different?

P

U| $\{a^m b^m c^m : m \geq 0\} \in P, \notin$ context-free

context-free

U| $\{a^m b^m : m \geq 0\} \in$ context-free, $\notin$ regular

regular

- Recall: P $\subseteq$ NP $\subseteq$ EXP
- Next Claim: P $\neq$ EXP
- So either P $\neq$ NP, or NP $\neq$ EXP
- We expect both to be true
- We can't prove any

- Claim: P ≠ EXP
- Proof: Consider D := "On input TM M

  run M on input M for $2^{|M|}$ steps

  if it accepts, REJECT

  otherwise, ACCEPT"

- L(D) $\in$ TIME(??)

- Claim: P ≠ EXP
- Proof: Consider D := "On input TM M
  run M on input M for $2^{|M|}$ steps
  if it accepts, REJECT
  otherwise, ACCEPT"

- L(D) $\in$ TIME($n\ 2^n$), so L(D) $\in$?

- To run M for 1 step, D takes at most $n = |M|$ steps

- This is a loose bound, sufficient for our purposes

- Claim: P ≠ EXP
- Proof: Consider D := "On input TM M

  run M on input M for $2^{|M|}$ steps

  if it accepts, REJECT

  otherwise, ACCEPT"

- L(D) $\in$ TIME($n\,2^n$), so L(D) $\in$ EXP

>

- Claim: P ≠ EXP
- Proof: Consider D := "On input TM M

  run M on input M for $2^{|M|}$ steps

  if it accepts, REJECT

  otherwise, ACCEPT"

- L(D) $\in$ TIME($n\, 2^n$), so L(D) $\in$ EXP
- We show L(D) $\notin$ P by contradiction:

- Claim: P ≠ EXP
- Proof: Consider D := "On input TM M

  run M on input M for $2^{|M|}$ steps

  if it accepts, REJECT

  otherwise, ACCEPT"

- L(D) $\in$ TIME($n\,2^n$), so L(D) $\in$ EXP
- We show L(D) $\notin$ P by contradiction: Assume L(D) $\in$ P

  Then $\exists$ TM N, integer c : L(N)=L(D), N runs in time n^c

  So N(N) = D(N) = ?

- Claim: P ≠ EXP
- Proof: Consider D := "On input TM M
  run M on input M for $2^{|M|}$ steps
  if it accepts, REJECT
  otherwise, ACCEPT"

- L(D) $\in$ TIME($n\, 2^n$), so L(D) $\in$ EXP
- We show L(D) $\notin$ P by contradiction: Assume L(D) $\in$ P

  Then $\exists$ TM N, integer c : L(N)=L(D), N runs in time n^c

  So N(N) = D(N) = not N(N), contradiction, so L(D) $\notin$ P

  ($n^c \leq 2^n$) Done

- Technical detail: Need $n^c \leq 2^n$ where $n = |N|$

- Since c is fixed, above true for sufficiently large n

- Need representation of programs where each program appears infinitely often

- This is true for every reasonable representation

- For example, add white spaces to your JAVA code

- Claim: $P \neq EXP$

- We have concluded the proof of this claim

- But the decidable language shown $\notin P$ is "unnatural"

- Next we use above claim to give a more natural one

- This will be similar to the proof that
  $\{G : G \text{ is CFG and } L(G) = \Sigma^* \}$ is undecidable

- Recall regular expressions

Definition Regular expressions RE over Σ are:

$\varnothing$

ε

a	if a in Σ
R R'	if R, R' are RE
R U R'	if R, R' are RE
R*	if R is RE

Example: $\Sigma^*aab\Sigma^*$, $(a^*ba^*ba^*)^*$

- All-RE = {R : R is RE and $L(R) = \Sigma^*$ }

- It is not known if All-RE $\in$ P

- We consider a more powerful type of RE, RE with exponentiation, abbreviated REE, then we prove All-REE $\notin$ P

- Definition:

  Regular expressions with exponentiation (REE)

∅	
ε	
a	if a in Σ
R R'	if R, R' are RE
R U R'	if R, R' are RE
R*	if R is RE
R^k	if R is RE

- $L(R^k) = L(R) \circ L(R) \circ \ldots \circ L(R)$ (k times)

- Note: In R^k, k is written in binary

- So $L(a^{1000000}) = \{ ? \}$

- Note: In R^k, k is written in binary
- So L($a^{1000000}$) =

  { aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

  aaaaaaaaaaaaaaaaaaaaaaa }
- This allows to write down compactly very long RE
- It is what makes the next problem hard

- Definition: All-REE = {R : R is REE and L(R) = Σ^* }

- Fact: All-REE is decidable
- Proof sketch:

  We already noted All-RE is decidable

  An REE can be converted to an RE. Done

- Theorem: All-REE $\notin$ P

- Theorem: All-REE ={R: R is REE and L(R)=∑*} $\notin$ P

- Proof: Suppose D decides All-REE in polynomial time

  We show EXP = P, violating previous theorem

>

- Theorem: All-REE ={R: R is REE and L(R)=∑*} $\notin$ P

- Proof: Suppose D decides All-REE in polynomial time
  We show EXP = P, violating previous theorem
- Let L $\in$ EXP. So $\exists$ c, TM M that decides L in time 2^{n^c}
- We construct D' that decides L in polynomial time:
- D' := "On input w:

  construct REE R : L(R) = ∑* ⇔ M accepts w

  then?

- Theorem: All-REE ={R: R is REE and L(R)=Σ^*} $\notin$ P

- Proof: Suppose D decides All-REE in polynomial time

  We show EXP = P, violating previous theorem

- Let L $\in$ EXP. So $\exists$ c, TM M that decides L in time 2^{n^c}
- We construct D' that decides L in polynomial time:
- D' := "On input w:

  construct REE R : L(R) = Σ^* $\Leftrightarrow$ M accepts w

  run D on R

  if it accepts, ACCEPT

  if it rejects, REJECT."

- Given M,c, and w, want $R : L(R) = \Sigma^* \Leftrightarrow$ M accepts w

  We construct $R : L(R) =$ all strings that are NOT rejecting computations of M on w

- Represent computation by sequence of configurations
  separated by #: $C_1 \# C_2 \# C_3 \ldots$

- Example: $q_0 000101 \# 1 q_3 00101 \# 10 q_2 0101$
- How many symbols in each configuration?

- Note: Because M runs in time 2^{n^c}
- On input w, |w| = n, M can only use ? tape cells

- Note: Because M runs in time 2^{n^c}
- On input w, |w| = n, M can only use 2^{n^c} tape cells
- Each of our configurations will have ≤ 2^{n^c} cells
- Different from proof that All-CF is undecidable?

- Note: Because M runs in time 2^{n^c}
- On input w, $|w| = n$, M can only use 2^{n^c} tape cells
- Each of our configurations will have exactly 2^{n^c} cells
- Different from proof that All-CF is undecidable: there we had no bound on the length of configurations

- Construct R: L(R) = all strings over $\Delta = \{\#\} \cup \Gamma \cup Q$ that are NOT rejecting computations of M on w

- A string $C_1\#C_2\#C_3\#...\#C_k$ is in L(R) $\Leftrightarrow$
  (a) C_1 is not the start configuration, or
  (b) C_k is not a reject configuration, or
  (c) $\exists i : C_i$ does not yield C_{i+1}

- We construct REE for (a), (b), and (c) separately then use closure under U

- (a) REE R_a : $L(R_a)$ = strings $C_1\#C_2\#C_3\#...\#C_k$ such that C_1 is not the start configuration q_0w

- $R_a = S_0 \cup S_1 \cup \ldots S_n \cup S_b \cup S_\#$

- S_0 = do not start with q_0 ?
- S_i = not w_i at position i, $1 \le i \le n$
- S_b = no _ in some position t, $n+2 \le t \le 2^{n^c}$

- $S_\#$ = no # in position $2^{n^c} + 1$

- (a) REE R_a : $L(R_a)$ = strings $C_1 \# C_2 \# C_3 \# ... \# C_k$ such that C_1 is not the start configuration $q_0 w$

- $R_a = S_0 \cup S_1 \cup \ldots S_n \cup S_b \cup S_\#$

- S_0 = do not start with q_0 $(\Delta - q_0)\,\Delta^*$
- S_i = not w_i at position i, $1 \le i \le n$?
- S_b = no _ in some position t, $n+2 \le t \le 2^{n^c}$

- $S_\#$ = no # in position $2^{n^c} + 1$

- (a) REE R_a : $L(R_a)$ = strings $C_1\#C_2\#C_3\#\ldots\#C_k$ such that C_1 is not the start configuration q_0w

- $R_a = S_0 \cup S_1 \cup \ldots S_n \cup S_b \cup S_\#$

- S_0 = do not start with q_0 $(\Delta - q_0)\,\Delta^*$
- S_i = not w_i at position i, $1 \le i \le n$ $\Delta^i\,(\Delta - w_i)\,\Delta^*$
- S_b = no _ in some position t, $n+2 \le t \le 2^{n^c}$

  ?

- $S_\#$ = no # in position $2^{n^c} + 1$

- (a) REE R_a : $L(R_a)$ = strings $C_1\#C_2\#C_3\#\ldots\#C_k$ such that C_1 is not the start configuration q_0w

- $R_a = S_0 \cup S_1 \cup \ldots S_n \cup S_b \cup S_\#$

- S_0 = do not start with q_0 $(\Delta - q_0)\,\Delta^*$
- S_i = not w_i at position i, $1 \le i \le n$ $\Delta^i\,(\Delta - w_i)\,\Delta^*$
- S_b = no _ in some position t, $n+2 \le t \le 2^{n^c}$

$$\Delta^{n+1}\,(\Delta \cup \varepsilon)^{2^{n^c}-n-2}\;(\Delta - \_\,)\;\Delta^*$$

- $S_\#$ = no # in position $2^{n^c} + 1$?

- (a) REE R_a : $L(R_a)$ = strings $C_1\#C_2\#C_3\#...\#C_k$ such that C_1 is not the start configuration $q_0 w$

- $R_a = S_0 \cup S_1 \cup \ldots S_n \cup S_b \cup S_\#$

- S_0 = do not start with q_0 $(\Delta - q_0)\, \Delta^*$
- S_i = not w_i at position i, $1 \le i \le n$ $\Delta^i\, (\Delta - w_i)\, \Delta^*$
- S_b = no _ in some position t, $n+2 \le t \le 2^{n^c}$

$$\Delta^{n+1}\, (\Delta \cup \varepsilon)^{2^{n^c}-n-2}\, (\Delta - \_)\, \Delta^*$$

- $S_\#$ = no # in position $2^{n^c} + 1$ $\Delta^{2^{n^c}}\, (\Delta - \#)\, \Delta^*$

- (b) REE R_b : $L(R_b)$ = strings $C_1\#C_2\#C_3\#...\#C_k$ such that R_k is not a reject configuration

- $R_b = (\Delta - q_{reject})^*$

- (c) REE R_c : $L(R_c)$ = strings $C_1\#C_2\#C_3\#...\#C_k$ such that $\exists\, i : C_i$ does not yield C_{i+1}

- Here we exploit ? of TM computation

- (c) REE R_c : $L(R_c)$ = strings $C_1\#C_2\#C_3\#...\#C_k$ such that $\exists\, i : C_i$ does not yield C_{i+1}

- Here we exploit locality of TM computation

- Fact: [Locality of TM computation]
  TM configuration C_i yields C_{i+1}
  $\Leftrightarrow \forall\, j$, the 6 symbols $(C_i)_j$, $(C_i)_{j+1}$, $(C_i)_{j+2}$, $(C_{i+1})_j$, $(C_{i+1})_{j+1}$, $(C_{i+1})_{j+2}$
  are consistent with TM transition function δ

- So what does it mean if C_i does **not** yield C_{i+1} ?

- (c) REE R_c : $L(R_c)$ = strings $C_1\#C_2\#C_3\#\ldots\#C_k$ such that $\exists i$: C_i does not yield C_{i+1}

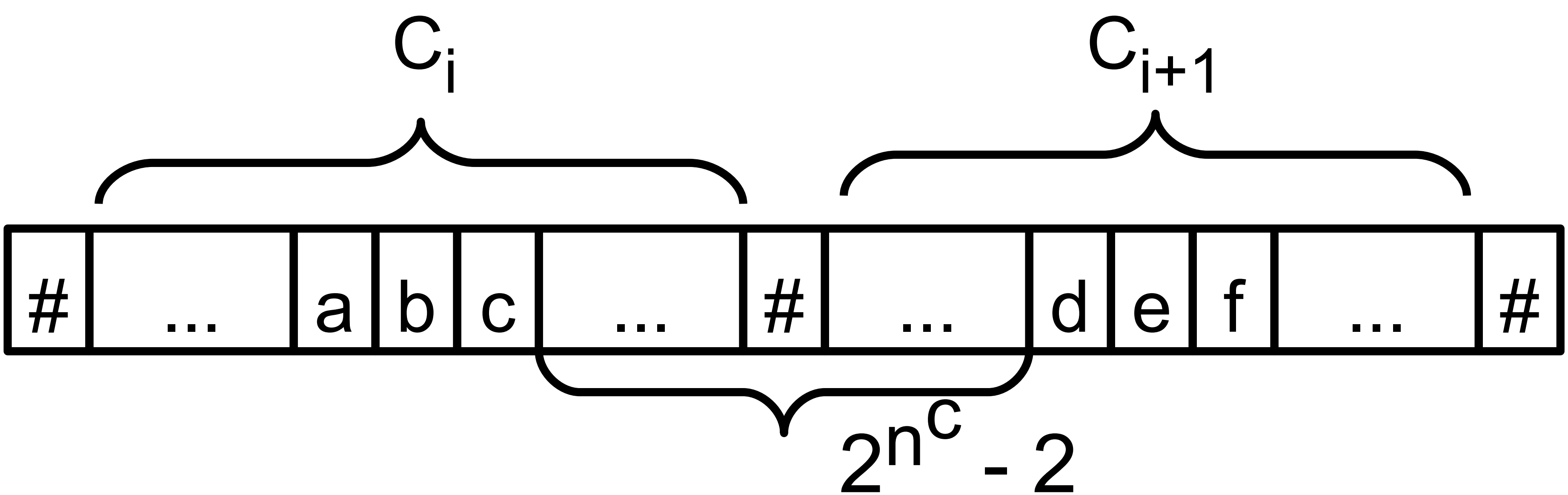
- Here we exploit locality of TM computation

- Fact: [Locality of TM computation]
TM configuration C_i does not yield C_{i+1}
$\Leftrightarrow \exists$ j , the 6 symbols $(C_i)_j$, $(C_i)_{j+1}$, $(C_i)_{j+2}$, $(C_{i+1})_j$, $(C_{i+1})_{j+1}$, $(C_{i+1})_{j+2}$
are not consistent with TM transition function δ

- (c) REE R_c : $L(R_c)$ = strings $C_1\#C_2\#C_3\#\ldots\#C_k$ such that $\exists i : C_i$ does not yield C_{i+1}

- $R_c = \bigcup \Delta^* \, abc \, \Delta^{(2^{n^c}-2)} \, def \, \Delta^*$

Union $\bigcup$ is over any $\begin{matrix} a\ b\ c \\ d\ e\ f \end{matrix}$ inconsistent with TM



- We also need that constructing R takes time polynomial in |w|

- Easily verified by looking at each piece

- For example:

$S_b = \quad \Delta^{n+1} \quad (\Delta \cup \varepsilon)^{2^{n^c}-n-2} (\Delta - \_) \ \Delta^*$

length $\leq$ $1 + \log(n+1) + \ 5 \ + \ n^c \ + \ 7$

$\leq n^{c+1}$

- Recap:

- Theorem: All-REE:={R: R is REE and L(G)=Σ^*} $\notin$ P

- But All-REE is decidable

- Key of proof is, given M, c, and w, construct REE R : L(R) = all strings that are NOT rejecting computations of M on w

- Use locality of TM computation (easier than JAVA)

- Theorem [Cook, Levin]: 3SAT $\in$ P $\Rightarrow$ P = NP
- Proof:

- Theorem [Cook, Levin]: 3SAT $\in$ P $\Rightarrow$ P = NP
- Proof:

Given M, w, and c, want to compute φ :

$\varphi \in \text{3SAT} \Leftrightarrow \underbrace{\exists y, |y| \leq |w|^c, M(w,y) \text{ accepts in time} \leq |w|^c}_{\text{Definition of NP}}$

- Computation of φ will run in polynomial time

- This proves the theorem because if 3SAT $\in$ P we can solve φ in polynomial-time

- Theorem [Cook, Levin]: 3SAT $\in$ P $\Rightarrow$ P = NP
- Proof:

Given M, w, and c, want to compute φ :

$\varphi \in$ 3SAT $\Leftrightarrow \exists y$, $|y| \leq |w|^c$, M(w,y) accepts in time $\leq |w|^c$

- It is convenient to let $k := |w|^c$

- Theorem [Cook, Levin]: 3SAT $\in$ P $\Rightarrow$ P = NP
- Proof:

Given M, w, and c, want to compute φ :

$\varphi \in$ 3SAT $\Leftrightarrow \exists$y, |y|$\le$ k, M(w,y) accepts in time $\le$ k

- Now use definition of accept

- Theorem [Cook, Levin]: 3SAT $\in$ P $\Rightarrow$ P = NP
- Proof:

Given M, w, and c, want to compute φ :

- $\varphi \in 3SAT \Leftrightarrow \exists y, |y| \leq k \;\; \exists C_1, C_2, \ldots, C_k$:

  C_1 is start configuration $q_0(w,y)$, AND

  C_k is accept configuration, AND

  $\forall i < k, \; C_i$ yields C_{i+1}

- Variables of φ are the symbols in y, $C_1, C_2, \ldots, C_k$

  encoded in binary (true/false)
- Example: $q_0 \rightarrow$ 001, ($\rightarrow$ 010

- Theorem [Cook, Levin]: 3SAT $\in$ P $\Rightarrow$ P = NP
- Proof:

Given M, w, and c, want to compute φ :

- $\varphi \in$ 3SAT $\Leftrightarrow$ $\exists$y, |y|$\leq$ k $\exists C_1, C_2, \ldots, C_k$:

C_1 is start configuration q_0(w,y), AND

C_k is accept configuration, AND

$\forall$ i < k, C_i yields C_{i+1}

- Variables of φ are the symbols in y, $C_1, C_2, \ldots, C_k$

Claim: For every i, $|C_i| \leq k$

Why?

- Theorem [Cook, Levin]: 3SAT $\in$ P $\Rightarrow$ P = NP
- Proof:

Given M, w, and c, want to compute φ :

- $\varphi \in$ 3SAT $\Leftrightarrow \exists y, |y| \le k$ $\exists C_1, |C_1| \le k, \ldots, C_k, |C_k| \le k$:

  C_1 is start configuration $q_0(w,y)$, AND

  C_k is accept configuration, AND

  $\forall i < k$, C_i yields C_{i+1}

- Variables of φ are the symbols in y, C_1, C_2, …, C_k

Claim: For every i, $|C_i| \le k$

- Because TM runs in time k, so uses $\le$ k tape cells

- Theorem [Cook, Levin]: 3SAT $\in$ P $\Rightarrow$ P = NP
- Proof:

Given M, w, and c, want to compute φ :

- $\varphi \in$ 3SAT $\Leftrightarrow \exists y, |y| \leq k \; \exists C_1, |C_1| \leq k, \ldots, C_k, |C_k| \leq k$:

  C_1 is start configuration $q_0(w,y)$, AND

  C_k is accept configuration, AND

  $\forall i < k$, C_i yields C_{i+1}

- Recall AND, ∀, are all the same as Λ used in SAT

- Theorem [Cook, Levin]: 3SAT $\in$ P $\Rightarrow$ P = NP
- Proof:

Given M, w, and c, want to compute φ :

- $\varphi \in$ 3SAT $\Leftrightarrow \exists y, |y| \le k \; \exists C_1, |C_1| \le k, \ldots, C_k, |C_k| \le k$:

  C_1 is start configuration $q_0(w,y) \wedge$

  C_k is accept configuration $\wedge$

  $\wedge_{i<k}$ C_i yields C_{i+1}

- Note $\wedge_{i<k}$ C_i yields C_{i+1} means

  C_1 yields $C_2 \wedge$ C_2 yields $C_3 \wedge \ldots \wedge$ C_{k-1} yields C_k

- Use ????? of TM computation to rewrite yield

- Theorem [Cook, Levin]: 3SAT $\in$ P $\Rightarrow$ P = NP
- Proof:

Given M, w, and c, want to compute φ :

- $\varphi \in$ 3SAT $\Leftrightarrow \exists y, |y| \leq k \ \exists C_1, |C_1| \leq k, \ldots, C_k, |C_k| \leq k$:

C_1 is start configuration $q_0(w,y) \wedge$

C_k is accept configuration $\wedge$

$\wedge_{i < k} \ C_i$ yields C_{i+1}

- Note $\wedge_{i < k} \ C_i$ yields C_{i+1} means

C_1 yields $C_2 \wedge C_2$ yields $C_3 \wedge \ldots \wedge C_{k-1}$ yields C_k

- Use locality of TM computation to rewrite yield

- Theorem [Cook, Levin]: 3SAT $\in$ P $\Rightarrow$ P = NP
- Proof:

Given M, w, and c, want to compute φ :

- $\varphi \in$ 3SAT $\Leftrightarrow \exists y, |y| \le k \ \exists C_1, |C_1| \le k, \ldots, C_k, |C_k| \le k$:

C_1 is start configuration $q_0(w,y) \wedge$

C_k is accept configuration $\wedge$

$\bigwedge_{i<k} \bigwedge_{j<k}$ $(C_i)_j, \ (C_i)_{j+1}, \ (C_i)_{j+2},$ $(C_{i+1})_j, (C_{i+1})_{j+1}, (C_{i+1})_{j+2}$

are consistent with TM

transition function

- Variables of φ = symbols in y, C_1, ..., C_k. What is φ ?

- Theorem [Cook, Levin]: $3SAT \in P \Rightarrow P = NP$
- Proof:

Given M, w, and c, want to compute φ :

- $\varphi \in 3SAT \Leftrightarrow \exists y, |y| \le k \; \exists C_1, |C_1| \le k, \ldots, C_k, |C_k| \le k$:

$\varphi =$

C_1 is start configuration $q_0(w,y) \wedge$
C_k is accept configuration $\wedge$
$\bigwedge_{i<k} \bigwedge_{j<k}$ $(C_i)_j, \; (C_i)_{j+1}, \; (C_i)_{j+2},$ $(C_{i+1})_j, (C_{i+1})_{j+1}, (C_{i+1})_{j+2}$ are consistent with TM transition function

With patience, easy to put this into 3SAT format Done

- Why do people believe that P ≠ NP?

- We have seen:
  Because otherwise problems in NP such as 3SAT, CLIQUE, etc. would be in P

- We will see:
  Because even many other tasks not known to be in NP would be in P!

- Theorem If P = NP there is a poly-time algorithm that given $\varphi \in$ SAT outputs a satisfying assignment
- Proof: ?

- Theorem If P = NP there is a poly-time algorithm that given $\varphi \in$ SAT outputs a satisfying assignment
- Proof: Let M be a poly-time machine deciding SAT.

  Define N := "On input φ:

  While there is a variable x in φ

  - Let φ_F be φ with x replaced with False.
  - If $M(\varphi_F) = 1$ then set x False,
  - otherwise set x True.

  Output the assignment."
- N(φ) runs in poly-time because it loops at most $|\varphi|$ times, and each time calls M which is poly-time

- Recall SAT = { φ : $\exists$ y φ(y) = true } $\in$ NP

- not SAT = { φ : $\forall$ y φ (y) = false }. Not known in NP

- Theorem If P = NP then not SAT $\in$ P
- Proof: ?

- Recall SAT = { φ : $\exists$ y φ(y) = true } $\in$ NP

- not SAT = { φ : $\forall$ y φ (y) = false }. Not known in NP

- Theorem If P = NP then not SAT $\in$ P
- Proof: Let M be a poly-time machine deciding SAT.

  Define N := "On input φ:

  Run M(φ)

  Return the opposite answer." ■

- P is closed under complement, NP is not known to be

Definition NTIME(t(n)) = { L : ∃ M : ∀ x of length n

x ∈ L ⟺ ∃ y, |y| ≤ t(n), M(x,y) accepts in ≤ t(n) steps}

Note NP = $\bigcup_c$ NTIME(n^c)

- Definition: NEXP = $\cup_c$ NTIME(2^{n^c})

- Theorem: P=NP ➔ EXP = NEXP

- Proof: Example of padding technique

Let L $\in$ NTIME(T(n)) where T(n) = 2^{n^c}

Let L' := { (x,$0^{T(n)}$) : x $\in$ L, |x| = n }

Note L' $\in$ NTIME(?

- Definition: NEXP = $\bigcup_c$ NTIME(2^{n^c})

- Theorem: P=NP ➔ EXP = NEXP

- Proof: Example of padding technique

Let L $\in$ NTIME(T(n)) where T(n) = 2^{n^c}

Let L' := { $(x,0^{T(n)})$: x $\in$ L, |x| = n }

Note L' $\in$ NTIME(n) $\subseteq$ NP = P. So let M solve L' in poly time.

EXP algorithm for L:

M' := "On input x;   ?

- Definition: NEXP = $\cup_c$ NTIME(2^{n^c})

- Theorem: P=NP ➔ EXP = NEXP

- Proof: Example of padding technique

Let L $\in$ NTIME(T(n)) where T(n) = 2^{n^c}

Let L' := { $(x,0^{T(n)})$: x $\in$ L, |x| = n }

Note L' $\in$ NTIME(n) $\subseteq$ NP = P. So let M solve L' in poly time.

EXP algorithm for L:

M' := "On input x; Replace x with $(x,0^{T(n)})$; Run M."

M'(x) = M$(x,0^{T(n)})$ = accept ⟺ x $\in$ L

M' runs in time 100 T(n). ■

- Padding:

Equivalences propagate “upward”

Intuition: if you have an equivalence between resources, then when you have even more of those resources the equivalence will continue to hold

Contrapositive of padding

Differences propagate “downward”

EXP ≠ NEXP ➔ P ≠ NP

NP $= \sum_1 P = \exists\, y : M(x,y) = 1$

co-NP $= \prod_1 P = \forall\, y : M(x,y) = 1$

$\sum_2 P = \exists\, y\ \forall\, z : M(x,y,z) = 1$

$\prod_2 P = \forall\, y\ \exists\, z : M(x,y,z) = 1$

$\sum_3 P = \exists\, y\ \forall\, z\ \exists\, w : M(x,y,z,w) = 1$

etc.

- Definition:

$\sum_i P$ = { L : $\exists$ poly-time M, polynomial q(n) :

$x \in L \Leftrightarrow \exists\, y_1 \in \{0,1\}^{q(n)}\ \forall y_2 \in \{0,1\}^{q(n)} \ldots Q\ y_{i+1} \in \{0,1\}^{q(n)}$

$M(x, y_1, y_2, \ldots, y_{i+1}) = 1\}$

$\prod_i P = \{L : \text{not } L \in \sum_i P\}$

same as swapping quantifiers above

Example MIN-F = { φ : $\forall\psi$: $|\psi|<|\varphi|, \exists x$: $\varphi(x) \neq \psi(x)$ }

MIN-F not known to be in NP

In which of the above classes is MIN-F?

- Definition:

$\sum_i P$ = { L : $\exists$ poly-time M, polynomial q(n) :

$x \in L \Leftrightarrow \exists y_1 \in \{0,1\}^{q(n)} \forall y_2 \in \{0,1\}^{q(n)} \ldots Q\, y_{i+1} \in \{0,1\}^{q(n)}$

$M(x, y_1, y_2, \ldots, y_{i+1}) = 1\}$

$\prod_i P = \{L : \text{not } L \in \sum_i P\}$

same as swapping quantifiers above

Example MIN-F = { φ : $\forall \psi$: $|\psi|<|\varphi|$, $\exists x$: $\varphi(x) \neq \psi(x)$ }

MIN-F not known to be in NP

In which of the above classes is MIN-F? $\prod_2 P$

Theorem: $P = NP \Rightarrow \Sigma_i P \cup \Pi_i P \subseteq P$

So if $P = NP$ then even MIN-F would be in P

Next we see the proof of the theorem

Theorem: $P = NP \Rightarrow \Sigma_i P \cup \Pi_i P \subseteq P$

Proof: By induction on i

Base case i = 1.

By assumption P = NP, recall Σ_1 = NP. So P = Σ_1 P.

Since P is closed under complement, it follows Π_1 = P.

Next we do the induction step.

We assume true for i and prove for i+1.

We will show Σ_{i+1} = P.

Result about Π_{i+1} follows again by complementing.

Theorem: $P = NP \Rightarrow \Sigma_i P \cup \Pi_i P \subseteq P$

Proof:

Let $L \in \Sigma_{i+1} P$, so $\exists$ poly-time M, polynomial q(n) :

$x \in L \Leftrightarrow \exists\, y_1 \in \{0,1\}^{q(n)}\ \forall\, y_2 \in \{0,1\}^{q(n)} \ldots Q\, y_{i+1} \in \{0,1\}^{q(n)}$

$M(x, y_1, y_2, \ldots, y_{i+1}) = 1$

Consider $L' := \{ (x, y_1) : \forall\, y_2 \in \{0,1\}^{q(n)} \ldots Q\, y_{i+1} \in \{0,1\}^{q(n)}$

$M(x, y_1, y_2, \ldots, y_{i+1}) = 1 \}$

$L' \in$?

Theorem: $P = NP \Rightarrow \sum_i P \cup \prod_i P \subseteq P$

Proof:

Let $L \in \sum_{i+1} P$, so $\exists$ poly-time M, polynomial q(n) :

$$x \in L \Leftrightarrow \exists\, y_1 \in \{0,1\}^{q(n)}\ \forall\, y_2 \in \{0,1\}^{q(n)} \ldots Q\, y_{i+1} \in \{0,1\}^{q(n)}$$

$$M(x, y_1, y_2, \ldots, y_{i+1}) = 1$$

Consider $L' := \{ (x, y_1) : \forall\, y_2 \in \{0,1\}^{q(n)} \ldots Q\, y_{i+1} \in \{0,1\}^{q(n)}$

$$M(x, y_1, y_2, \ldots, y_{i+1}) = 1 \}$$

$L' \in \prod_i P$. By induction hypothesis $L' \in P$.

So let poly-time machine M' solve L'.

So $x \in L \Leftrightarrow \exists\, y_1 \in \{0,1\}^{q(n)} : M'(x, y_1) = 1$

And so $L \in$?

Theorem: $P = NP \Rightarrow \Sigma_i P \cup \Pi_i P \subseteq P$

Proof:

Let $L \in \Sigma_{i+1} P$, so $\exists$ poly-time M, polynomial q(n) :

$$x \in L \Leftrightarrow \exists y_1 \in \{0,1\}^{q(n)} \forall y_2 \in \{0,1\}^{q(n)} \ldots Q\, y_{i+1} \in \{0,1\}^{q(n)}$$

$$M(x, y_1, y_2, \ldots, y_{i+1}) = 1$$

Consider $L' := \{ (x, y_1) : \forall y_2 \in \{0,1\}^{q(n)} \ldots Q\, y_{i+1} \in \{0,1\}^{q(n)}$

$$M(x, y_1, y_2, \ldots, y_{i+1}) = 1 \}$$

$L' \in \Pi_i P$. By induction hypothesis $L' \in P$.

So let poly-time machine M' solve L'.

So $x \in L \Leftrightarrow \exists y_1 \in \{0,1\}^{q(n)} : M'(x, y_1) = 1$

And so $L \in NP \Rightarrow L \in P$ ■

Randomized Complexity Classes

So far, we thought of “easy” as P

In fact, people think of “easy” as P + randomness

A model without randomness is out-of-date

An extra benefit is practicing probability theory which is fundamental to almost everything nowadays.

- We allow TM to toss coins/throw dice etc.

We write M(x,R) for output of M on input x, coin tosses R

- Definition: $L \in RP \iff \exists$ Turing machine M :

  $x \in L \Rightarrow Pr_R [M(x,R)=1] \geq 1/2$

  $x \notin L \Rightarrow Pr_R [M(x,R)=1] = 0$

  M(x,R) runs in time polynomial in |x|

- NP is same as RP with "$\geq 1/2$" replaced by ?

- Claim: $RP \subseteq NP$

- We allow TM to toss coins/throw dice etc.

We write M(x,R) for output of M on input x, coin tosses R

- Definition: $L \in RP \iff \exists$ Turing machine M :

  $x \in L \Rightarrow Pr_R\,[M(x,R)=1] \geq 1/2$

  $x \notin L \Rightarrow Pr_R\,[M(x,R)=1] = 0$

  M(x,R) runs in time polynomial in |x|

- NP is same as RP with "$\geq 1/2$" replaced by ">0"

- Claim: $RP \subseteq NP$

- Definition: $L \in RP \iff \exists$ Turing machine M :

  $x \in L \Rightarrow Pr_R [M(x,R)=1] \geq 1/2$; $x \notin L \Rightarrow Pr_R [M(x,R)=1] = 0$

  M(x,R) runs in time polynomial in |x|

- Claim: Definition is the same if we replace 1/2 with

  $1/n^c$, or $1 - 1/2^m$ for $m = n^c$, for any c

- Proof:

- Definition: $L \in RP \iff \exists$ Turing machine M :

  $x \in L \Rightarrow Pr_R [M(x,R)=1] \geq 1/2$; $x \notin L \Rightarrow Pr_R [M(x,R)=1] = 0$

  M(x,R) runs in time polynomial in |x|

- Claim: Definition is the same if we replace 1/2 with

  $1/n^c$, or $1 - 1/2^m$ for $m = n^c$, for any c

- Proof: Suppose $x \in L \Rightarrow Pr_R [M(x,R)=1] \geq 1/n^c$

Consider M' which …?

- Definition: $L \in RP$ <=> $\exists$ Turing machine M :

  $x \in L \Rightarrow Pr_R [M(x,R)=1] \geq 1/2$; $x \notin L \Rightarrow Pr_R [M(x,R)=1] = 0$

  M(x,R) runs in time polynomial in |x|

- Claim: Definition is the same if we replace 1/2 with

  $1/n^c$, or $1 - 1/2^m$ for $m = n^c$, for any c

- Proof: Suppose $x \in L \Rightarrow Pr_R [M(x,R)=1] \geq 1/n^c$

Consider M' which runs M t times independently, outputs OR.

$x \notin L \Rightarrow Pr_R [M'(x,R)=1] = ?$

- Definition: $L \in RP \iff \exists$ Turing machine M :

  $x \in L \Rightarrow Pr_R [M(x,R)=1] \geq 1/2$; $x \notin L \Rightarrow Pr_R [M(x,R)=1] = 0$

  M(x,R) runs in time polynomial in |x|

- Claim: Definition is the same if we replace 1/2 with

  $1/n^c$, or $1 - 1/2^m$ for $m = n^c$, for any c

- Proof: Suppose $x \in L \Rightarrow Pr_R [M(x,R)=1] \geq 1/n^c$

Consider M' which runs M t times independently, outputs OR.

$x \notin L \Rightarrow Pr_R [M'(x,R)=1] = 0$

$x \in L \Rightarrow Pr_R [M'(x,R)=0] =?$

- Definition: $L \in RP \Leftrightarrow \exists$ Turing machine M :

  $x \in L \Rightarrow Pr_R [M(x,R)=1] \geq 1/2; \;\; x \notin L \Rightarrow Pr_R [M(x,R)=1] = 0$

  M(x,R) runs in time polynomial in |x|

- Claim: Definition is the same if we replace 1/2 with

  $1/n^c$, or $1 - 1/2^m$ for $m = n^c$, for any c

- Proof: Suppose $x \in L \Rightarrow Pr_R [M(x,R)=1] \geq 1/n^c$

Consider M' which runs M t times independently, outputs OR.

$x \notin L \Rightarrow Pr_R [M' (x,R)=1] = 0$

$x \in L \Rightarrow Pr_R [M' (x,R)=0] = (Pr_R [M (x,R)=0])^t \leq ?$

- Definition: $L \in RP \iff \exists$ Turing machine M :

  $x \in L \Rightarrow Pr_R\,[M(x,R)=1] \geq 1/2$; $x \notin L \Rightarrow Pr_R\,[M(x,R)=1] = 0$

  M(x,R) runs in time polynomial in |x|

- Claim: Definition is the same if we replace 1/2 with

  $1/n^c$, or $1 - 1/2^m$ for $m = n^c$, for any c

- Proof: Suppose $x \in L \Rightarrow Pr_R\,[M(x,R)=1] \geq 1/n^c$

Consider M' which runs M t times independently, outputs OR.

$x \notin L \Rightarrow Pr_R\,[M'(x,R)=1] = 0$

$x \in L \Rightarrow Pr_R\,[M'(x,R)=0] = (Pr_R\,[M(x,R)=0])^t \leq (1 - 1/n^c)^t \leq ?$

- **Definition:** $L \in RP \iff \exists$ Turing machine M :

  $x \in L \Rightarrow Pr_R[M(x,R)=1] \geq 1/2$; $x \notin L \Rightarrow Pr_R[M(x,R)=1] = 0$

  M(x,R) runs in time polynomial in |x|

- **Claim:** Definition is the same if we replace 1/2 with

  $1/n^c$, or $1 - 1/2^m$ for $m = n^c$, for any c

- **Proof:** Suppose $x \in L \Rightarrow Pr_R[M(x,R)=1] \geq 1/n^c$

Consider M' which runs M t times independently, outputs OR.

$x \notin L \Rightarrow Pr_R[M'(x,R)=1] = 0$

$x \in L \Rightarrow Pr_R[M'(x,R)=0] = (Pr_R[M(x,R)=0])^t \leq (1 - 1/n^c)^t \leq 1/2^m$

for $t = n^d$ for d a constant dependent on c.

Running time of M' = ?

- Definition: $L \in RP$ <=> $\exists$ Turing machine M :

  $x \in L \Rightarrow Pr_R[M(x,R)=1] \geq 1/2$; $x \notin L \Rightarrow Pr_R[M(x,R)=1] = 0$

  $M(x,R)$ runs in time polynomial in $|x|$

- Claim: Definition is the same if we replace 1/2 with

  $1/n^c$, or $1 - 1/2^m$ for $m = n^c$, for any c

- Proof: Suppose $x \in L \Rightarrow Pr_R[M(x,R)=1] \geq 1/n^c$

Consider M' which runs M t times independently, outputs OR.

$x \notin L \Rightarrow Pr_R[M'(x,R)=1] = 0$

$x \in L \Rightarrow Pr_R[M'(x,R)=0] = (Pr_R[M(x,R)=0])^t \leq (1 - 1/n^c)^t \leq 1/2^m$

for $t = n^d$ for d a constant dependent on c.

Running time of M' = t x running time of M = polynomial ■

ZPP can be equivalently defined as the set of L such that:

1) $L \in RP$, not $L \in RP$

2) There is a machine M for L :

$\forall$ x, $\forall$ R, $M(x,R) \in \{L(x), ?\}$,

$\forall$ x, $Pr_R\ [M(x,R) = ?\ ] \leq 1/2$

M(x,R) runs in time polynomial in |x|

3) There is a machine M for L : $\forall$ x, $\forall$ R, $M(x,R) = L(x)$

the expected running time of M on x is poly(n)

- Definition: $L \in BPP \iff \exists$ Turing machine M :

  $x \in L \Rightarrow Pr_R [M(x,R)=1] \geq 2/3$

  $x \notin L \Rightarrow Pr_R [M(x,R)=1] \leq 1/3$

  M(x,R) runs in time polynomial in |x|

- Not known if $BPP \subseteq NP$

- Claim: Definition is the same if we replace (2/3,1/3) with

  $(1/2+1/n^c , 1/2-1/n^c)$, or $(1 - 1/2^m , 1/2^m)$

- Proof sketch: Consider M' which runs M t times

independently, outputs ???????

- Definition: $L \in BPP \iff \exists$ Turing machine M :

  $x \in L \Rightarrow Pr_R [M(x,R)=1] \geq 2/3$

  $x \notin L \Rightarrow Pr_R [M(x,R)=1] \leq 1/3$

  M(x,R) runs in time polynomial in |x|

- Not known if $BPP \subseteq NP$

- Claim: Definition is the same if we replace (2/3,1/3) with

  $(1/2+1/n^c , 1/2-1/n^c)$, or $(1 - 1/2^m , 1/2^m)$

- Proof sketch: Consider M' which runs M t times

independently, outputs MAJORITY ■

- Claim: $P \subseteq ZPP \subseteq RP \subseteq BPP$
- Proof: By definition. ■

- Big open question, is P = ZPP = RP = BPP?

Surprisingly, this is believed to be the case

- Recall: $RP \subseteq NP$, but BPP not known to be in NP.

- Claim: BPP $\subseteq \Sigma_2 P$

- Claim: BPP $\subseteq \Sigma_2$ P
- Proof: Let M(x,R) toss |R| = r coins, and have error < $1/r^2$

Fix x and ask: Can we cover $\{0,1\}^r$ with r shifts of

$$A := \{ R \in \{0,1\}^r : M(x,R) = 1 \} ?$$

For s $\in \{0,1\}^r$, the s-shift is s+A := { s XOR a : a $\in$ A } $\subseteq \{0,1\}^r$

We'll show the answer to this question is equivalent to x $\in$ L

That concludes the proof because ?

How do you conclude that L(M) is in Σ_2 P ?

- Claim: BPP $\subseteq \Sigma_2$ P
- Proof: Let M(x,R) toss |R| = r coins, and have error < $1/r^2$

Fix x and ask: Can we cover $\{0,1\}^r$ with r shifts of

$$A := \{ R \in \{0,1\}^r : M(x,R) = 1 \} ?$$

For $s \in \{0,1\}^r$, the s-shift is s+A := { s XOR a : a $\in$ A } $\subseteq \{0,1\}^r$

We'll show the answer to this question is equivalent to x $\in$ L

That concludes the proof because we have

M(x,R) = 1 $\Leftrightarrow \exists s_1, \ldots, s_r : \forall y \in \{0,1\}^r, y \in \cup_r s_r + A$

$\Leftrightarrow \exists s_1, \ldots, s_r : \forall y \in \{0,1\}^r$, ...???????????????????

- Claim: BPP $\subseteq \Sigma_2$ P
- Proof: Let M(x,R) toss |R| = r coins, and have error < $1/r^2$

Fix x and ask: Can we cover $\{0,1\}^r$ with r shifts of

$$A := \{ R \in \{0,1\}^r : M(x,R) = 1 \} ?$$

For s $\in \{0,1\}^r$, the s-shift is s+A := { s XOR a : a $\in$ A } $\subseteq \{0,1\}^r$

We'll show the answer to this question is equivalent to x $\in$ L

That concludes the proof because we have

$M(x,R) = 1 \iff \exists\, s_1, \ldots, s_r : \forall\, y \in \{0,1\}^r ,\; y \in \cup_r s_r + A$

$\iff \exists\, s_1, \ldots, s_r : \forall\, y \in \{0,1\}^r,\; \vee_{i=1}^{r}\; M(x, y + s_i) = 1$

- Claim: BPP $\subseteq \Sigma_2$ P

- Proof: Let M(x,R) toss |R| = r coins, and have error < $1/r^2$

Fix x and ask: Can we cover $\{0,1\}^r$ with r shifts of

$$A := \{ R \in \{0,1\}^r : M(x,R) = 1 \} ?$$

For s $\in \{0,1\}^r$, the s-shift is s+A := { s XOR a : a $\in$ A } $\subseteq \{0,1\}^r$

- x $\notin$ L, we show we cannot cover. Note |A| <= ?

- Claim: BPP $\subseteq \Sigma_2$ P

- Proof: Let M(x,R) toss |R| = r coins, and have error < $1/r^2$

Fix x and ask: Can we cover $\{0,1\}^r$ with r shifts of

$$A := \{ R \in \{0,1\}^r : M(x,R) = 1 \} ?$$

For $s \in \{0,1\}^r$, the s-shift is $s+A := \{ s \text{ XOR } a : a \in A \} \subseteq \{0,1\}^r$

- $x \notin L$, we show we cannot cover. Note $|A| <= 2^r / r^2$.

$\forall s_1, \ldots, s_r : |s_1+A \cup s_2+A \cup \ldots \cup s_r+A| \leq$?

- Claim: BPP $\subseteq \Sigma_2$ P

- Proof: Let M(x,R) toss |R| = r coins, and have error < $1/r^2$

Fix x and ask: Can we cover $\{0,1\}^r$ with r shifts of

$$A := \{ R \in \{0,1\}^r : M(x,R) = 1 \} ?$$

For s $\in \{0,1\}^r$, the s-shift is s+A := { s XOR a : a $\in$ A } $\subseteq \{0,1\}^r$

- x $\notin$ L, we show we cannot cover. Note |A| <= $2^r / r^2$.

$\forall$ s_1, …, s_r : $|s_1+A \cup s_2+A \cup \ldots \cup s_r+A| \leq r\,|A| \leq$?

- Claim: BPP $\subseteq \Sigma_2$ P

- Proof: Let M(x,R) toss |R| = r coins, and have error < $1/r^2$

Fix x and ask: Can we cover $\{0,1\}^r$ with r shifts of

$$A := \{ R \in \{0,1\}^r : M(x,R) = 1 \} ?$$

For $s \in \{0,1\}^r$, the s-shift is $s+A := \{ s \text{ XOR } a : a \in A \} \subseteq \{0,1\}^r$

- x $\notin$ L, we show we cannot cover. Note $|A| <= 2^r / r^2$.

$\forall s_1, \ldots, s_r : |s_1+A \cup s_2+A \cup \ldots \cup s_r+A| \le r|A| \le r\, 2^r / r^2 < 2^r$

- x $\in$ L, we show we can cover.

Idea pick the shifts at random and show Pr[do not cover] < ?

- Claim: BPP $\subseteq \Sigma_2$ P
- Proof: Let M(x,R) toss |R| = r coins, and have error < $1/r^2$

Fix x and ask: Can we cover $\{0,1\}^r$ with r shifts of

$$A := \{ R \in \{0,1\}^r : M(x,R) = 1 \} ?$$

For $s \in \{0,1\}^r$, the s-shift is s+A := { s XOR a : a $\in$ A } $\subseteq \{0,1\}^r$

- x $\notin$ L, we show we cannot cover. Note $|A| <= 2^r / r^2$.

$\forall s_1, \ldots, s_r : |s_1+A \cup s_2+A \cup \ldots \cup s_r+A| \le r|A| \le r\, 2^r / r^2 < 2^r$

- x $\in$ L, we show we can cover.

Idea pick the shifts at random and show Pr[do not cover] < 1:

$Pr_{s1, \ldots, sr} [\exists y \in \{0,1\}^r : y \notin \cup_r s_r + A] \le$

?

- Claim: BPP $\subseteq \Sigma_2$ P

- Proof: Let M(x,R) toss |R| = r coins, and have error < $1/r^2$

Fix x and ask: Can we cover $\{0,1\}^r$ with r shifts of

$$A := \{ R \in \{0,1\}^r : M(x,R) = 1 \} ?$$

For s $\in \{0,1\}^r$, the s-shift is s+A := { s XOR a : a $\in$ A } $\subseteq \{0,1\}^r$

- x $\notin$ L, we show we cannot cover. Note $|A| <= 2^r / r^2$.

$\forall s_1, \ldots, s_r : |s_1+A \cup s_2+A \cup \ldots \cup s_r+A| \le r|A| \le r 2^r / r^2 < 2^r$

- x $\in$ L, we show we can cover.

Idea pick the shifts at random and show Pr[do not cover] < 1:

$Pr_{s1, \ldots, sr} [\exists y \in \{0,1\}^r : y \notin U_r\, s_r + A] \le$

$\Sigma_y Pr_{s1,\ldots,sr}[y \notin U_r\, s_r + A] = ?$

- Claim: BPP $\subseteq \Sigma_2$ P

- Proof: Let M(x,R) toss |R| = r coins, and have error < $1/r^2$

Fix x and ask: Can we cover $\{0,1\}^r$ with r shifts of

$$A := \{ R \in \{0,1\}^r : M(x,R) = 1 \} ?$$

For $s \in \{0,1\}^r$, the s-shift is s+A := { s XOR a : $a \in A$ } $\subseteq \{0,1\}^r$

- $x \notin L$, we show we cannot cover. Note $|A| \le 2^r / r^2$.

$\forall s_1, \ldots, s_r : |s_1+A \cup s_2+A \cup \ldots \cup s_r+A| \le r|A| \le r\, 2^r / r^2 < 2^r$

- $x \in L$, we show we can cover.

Idea pick the shifts at random and show Pr[do not cover] < 1:

$Pr_{s1, \ldots, sr}[\exists y \in \{0,1\}^r : y \notin U_r\, s_r + A] \le$

$\sum_y Pr_{s1,\ldots,sr}[y \notin U_r\, s_r + A] = \sum_y (Pr_s[y \notin s + A])^r \le$?

- Claim: $BPP \subseteq \Sigma_2 P$
- Proof: Let $M(x,R)$ toss $|R| = r$ coins, and have error $< 1/r^2$

Fix x and ask: Can we cover $\{0,1\}^r$ with r shifts of

$$A := \{ R \in \{0,1\}^r : M(x,R) = 1 \} ?$$

For $s \in \{0,1\}^r$, the s-shift is $s+A := \{ s \text{ XOR } a : a \in A \} \subseteq \{0,1\}^r$

- $x \notin L$, we show we cannot cover. Note $|A| <= 2^r / r^2$.

$\forall s_1, \ldots, s_r : |s_1+A \cup s_2+A \cup \ldots \cup s_r+A| \leq r\,|A| \leq r\,2^r / r^2 < 2^r$

- $x \in L$, we show we can cover.

Idea pick the shifts at random and show Pr[do not cover] < 1:

$Pr_{s1, \ldots, sr} [\exists y \in \{0,1\}^r : y \notin \cup_r s_r + A] \leq$

$\sum_y Pr_{s1,\ldots,sr}[y \notin \cup_r s_r + A] = \sum_y (Pr_s[y \notin s + A])^r \leq \sum_y (1/r^2)^r < 1$

■

- Corollary: P = NP => P = BPP.

- Proof:

  ?

- Corollary: P = NP => P = BPP.

- Proof:

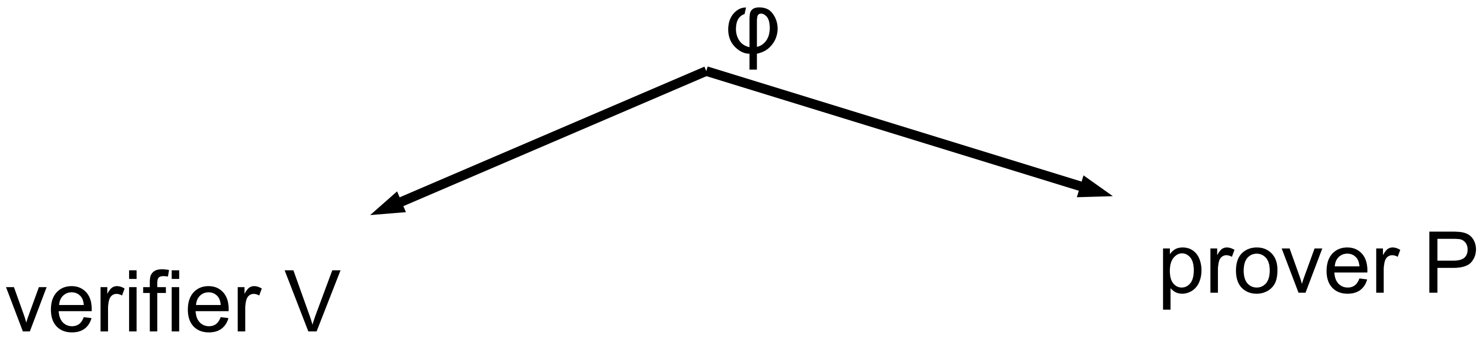
  P = NP => P = PH, and so

  $P \subseteq BPP \subseteq PH = P$ ■

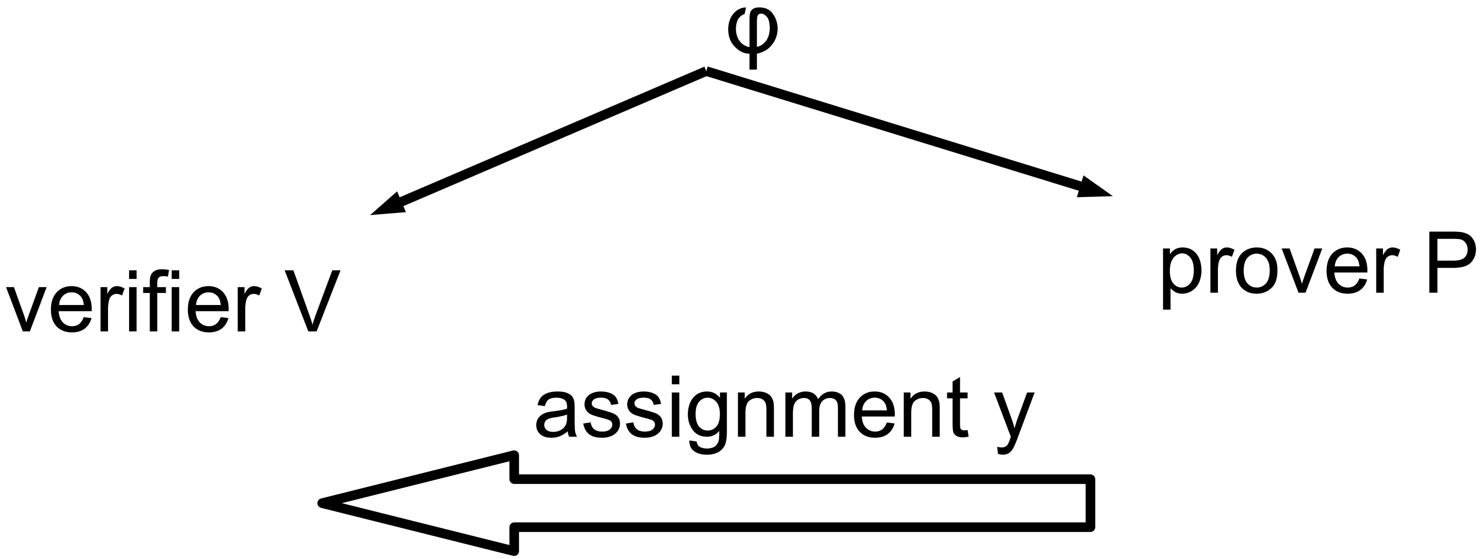
Interactive Proof Systems

- NP as a “proof system”
- If L $\in$ NP, we can think of
- a polynomial-time verifier V, and
- an all-powerful prover P.
- They are both given input w.
- P needs to convince V that w $\in$ L

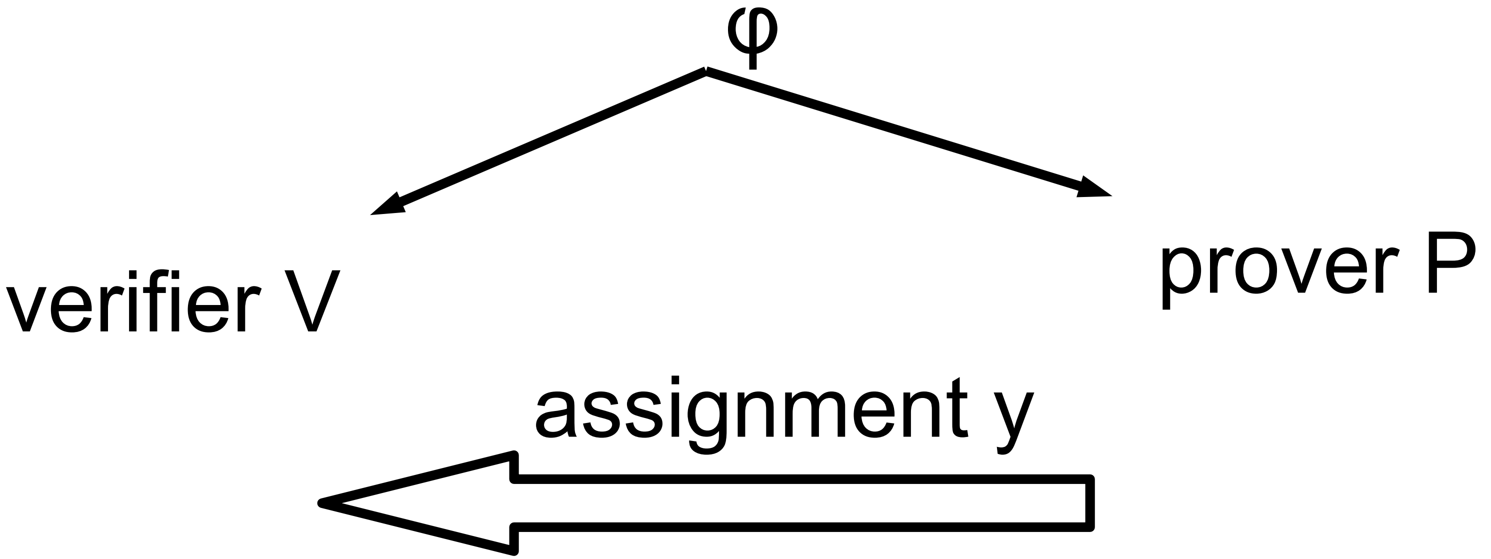
• Example: Proof system for SAT



- Example: Proof system for SAT



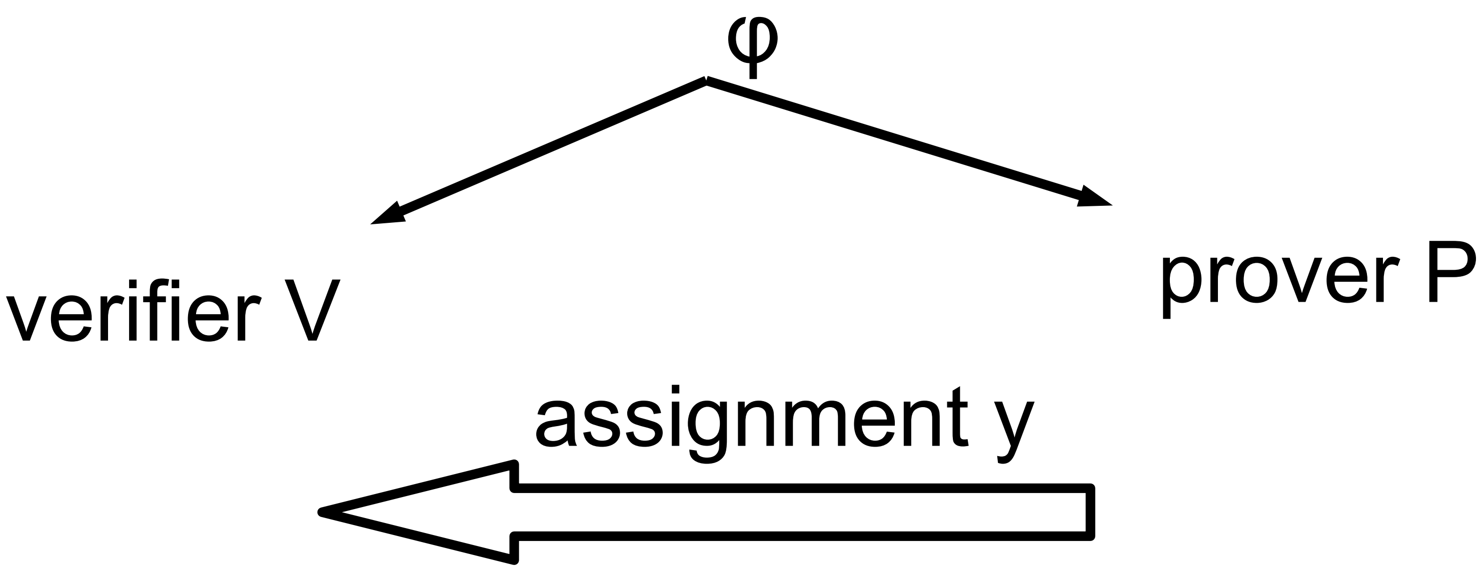
- Example: Proof system for SAT



V accepts

if y satisfies φ

- Example: Proof system for SAT



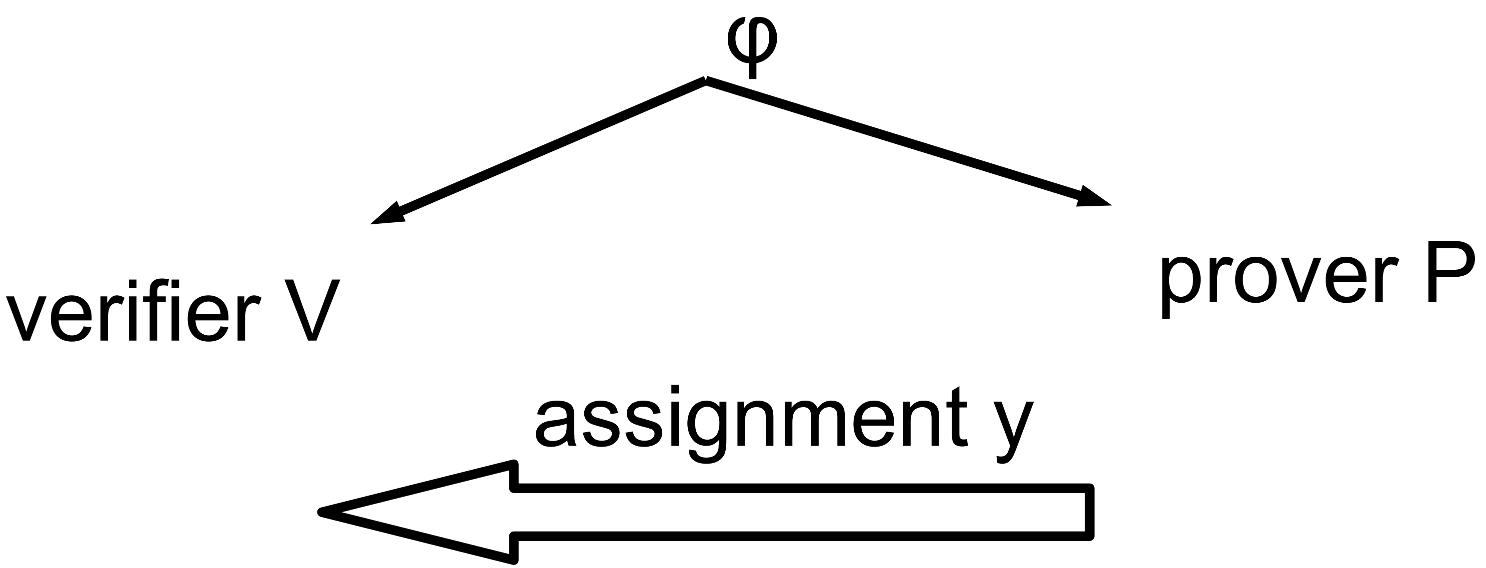
V accepts

if y satisfies φ

If $\varphi \in SAT$, there exists P that makes V accept:

P simply sends a satisfying assignment y

- Example: Proof system for SAT



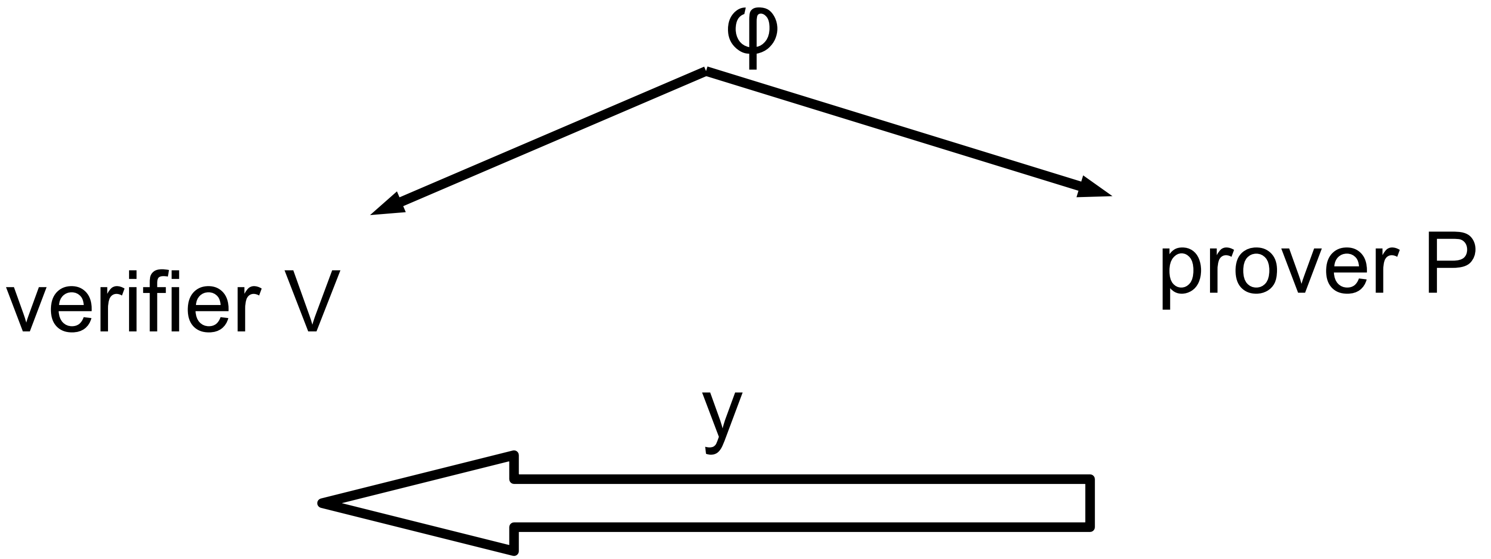
V accepts

if y satisfies φ

If $\varphi \notin$ SAT, then no P makes V accept:

whatever P sends, V will not accept

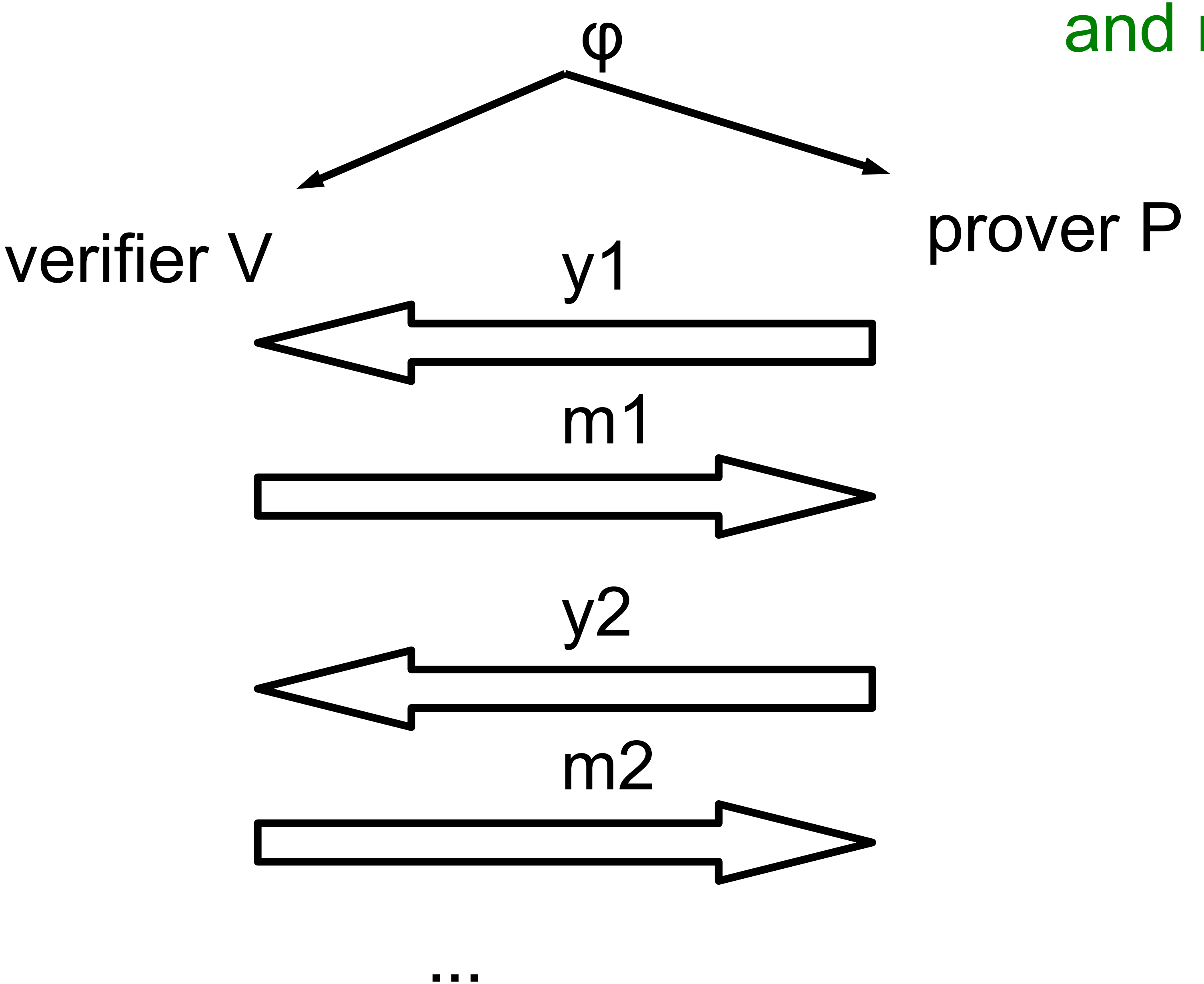
• Open question: Proof system for not SAT?



Can a prover send some y that convinces V that φ is not satisfiable?

Believed to be impossible.

• Fact: ∃ Proof system for not SAT, with interaction and randomization



V accepts with high probability ⇔ $\varphi \notin SAT$

- Fact: ∃ Proof system for not SAT, with interaction

randomization

φ

verifier V

Interaction is powerful!

That is why you come to class

Previous result has two components:

- Interaction
- Randomization

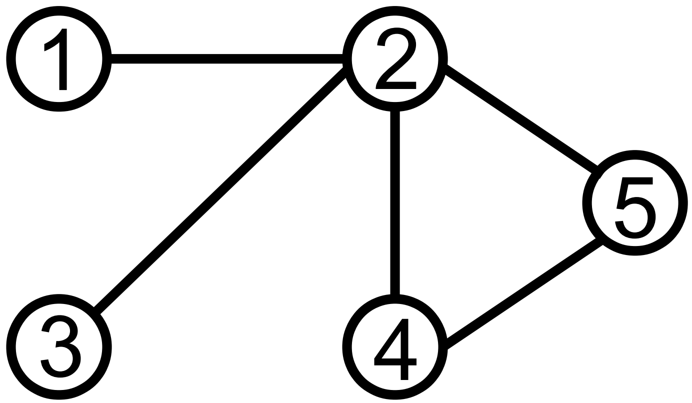
Note every computation has some error probability:

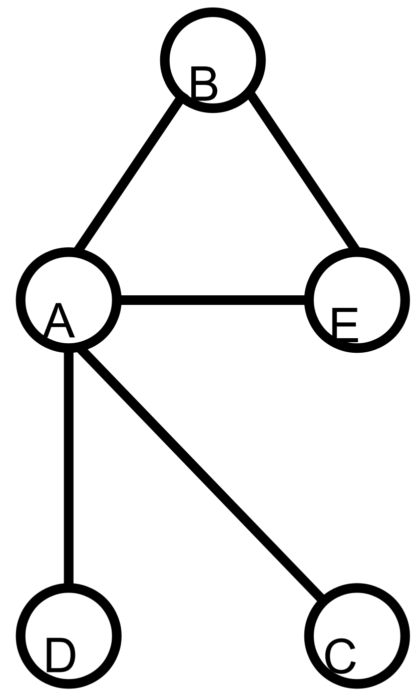
There is always a chance an asteroid hits my pc

The error in previous result is just as small

Graph Labeling

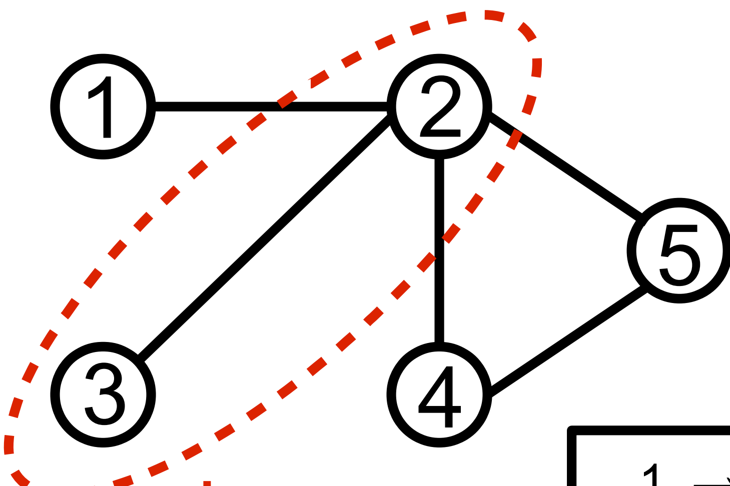
Two graphs are label-equivalent if labels of one can be mapped to the other while preserving the structure.





Graph Labeling

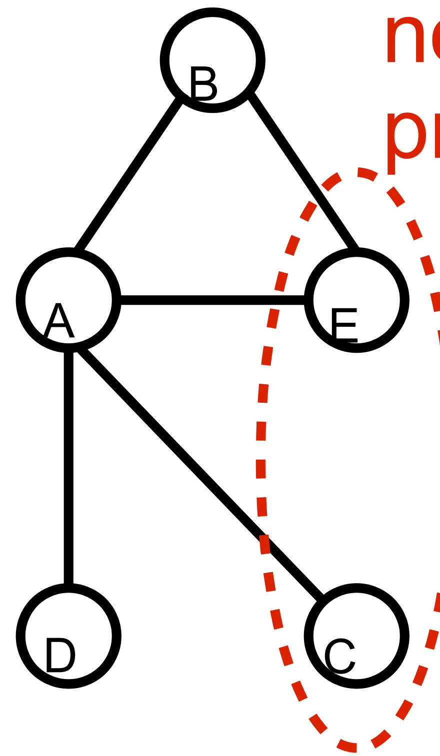
Two graphs are label-equivalent if labels of one can be mapped to the other while preserving the structure.



edge present

1 → A
2 → E
3 → C
4 → D
5 → B

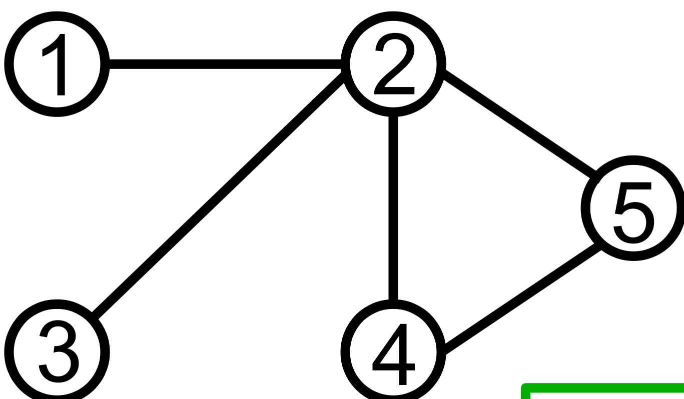
This map does NOT preserve the structure



no edge present

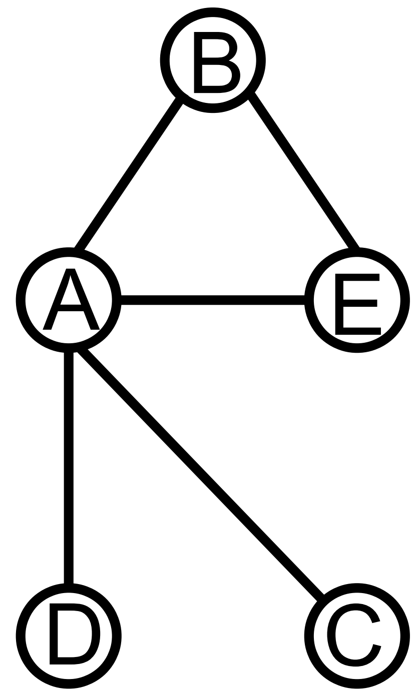
Graph Labeling

Two graphs are label-equivalent if labels of one can be mapped to the other while preserving the structure.



This map DOES preserve the structure

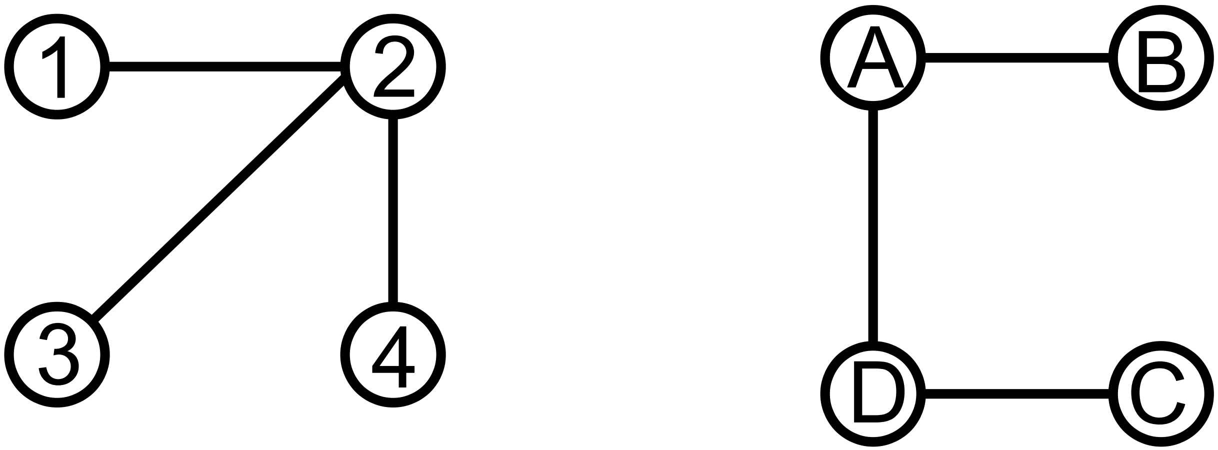
1 → D
2 → A
3 → C
4 → E
5 → B



So, these graphs are label-equivalent

Graph Labeling

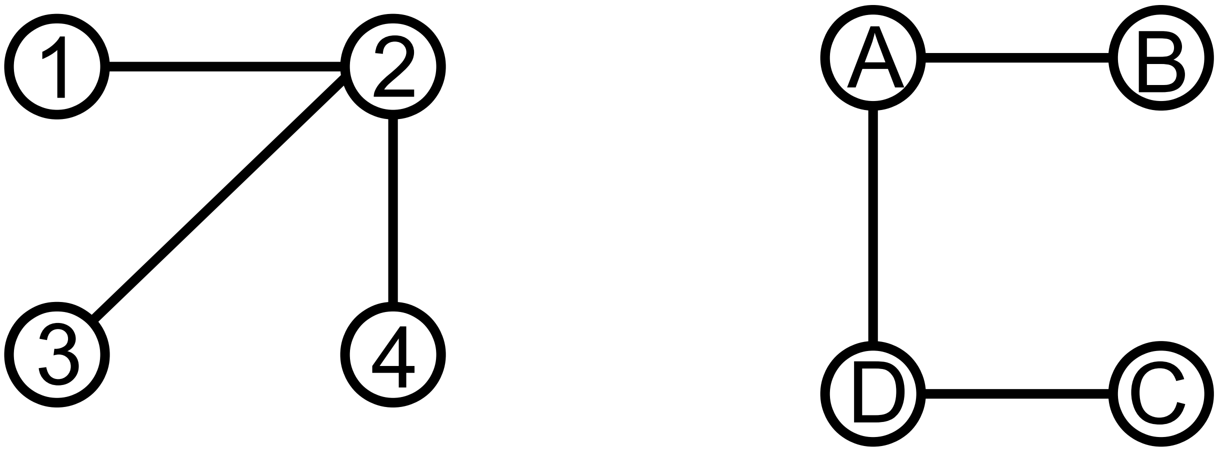
Two graphs are label-equivalent if labels of one can be mapped to the other while preserving the structure.



These graphs are ??? label-equivalent:

Graph Labeling

Two graphs are label-equivalent if labels of one can be mapped to the other while preserving the structure.

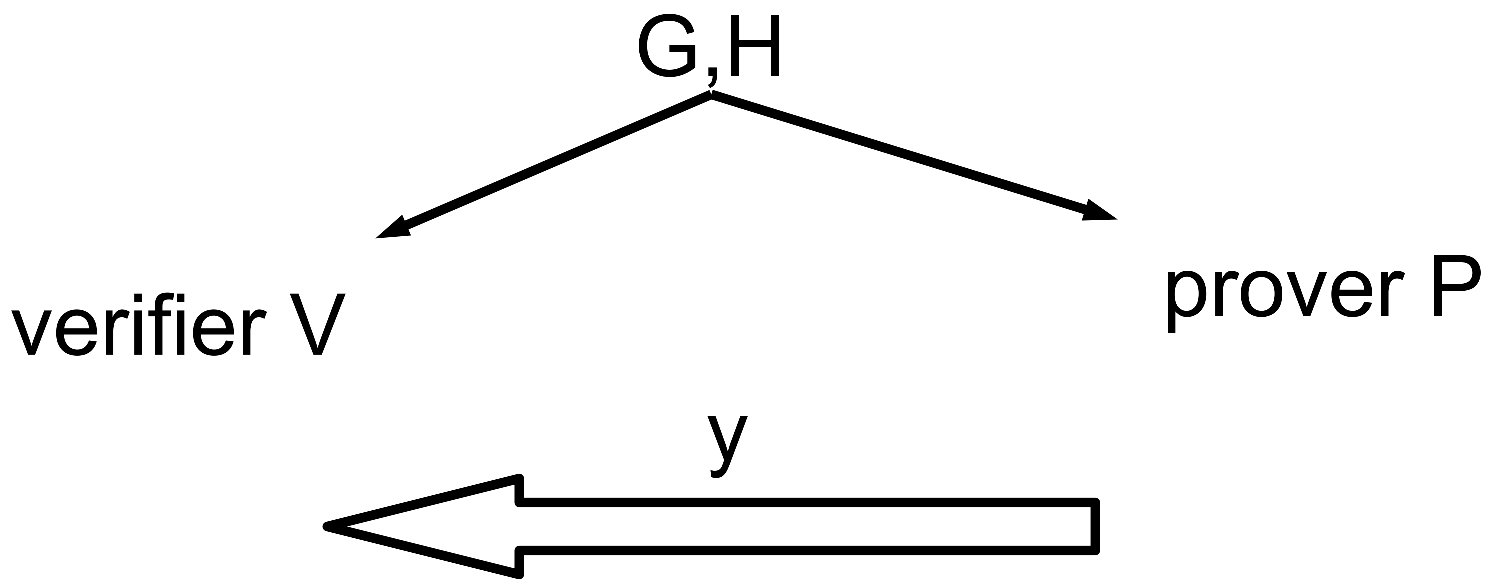


These graphs are NOT label-equivalent:

- A,B,C,D each touch two or fewer edges
- 2 touches three edges.

- LABEL-NEQ = {(G,H) | G and H are graphs that are **not** label-equivalent}

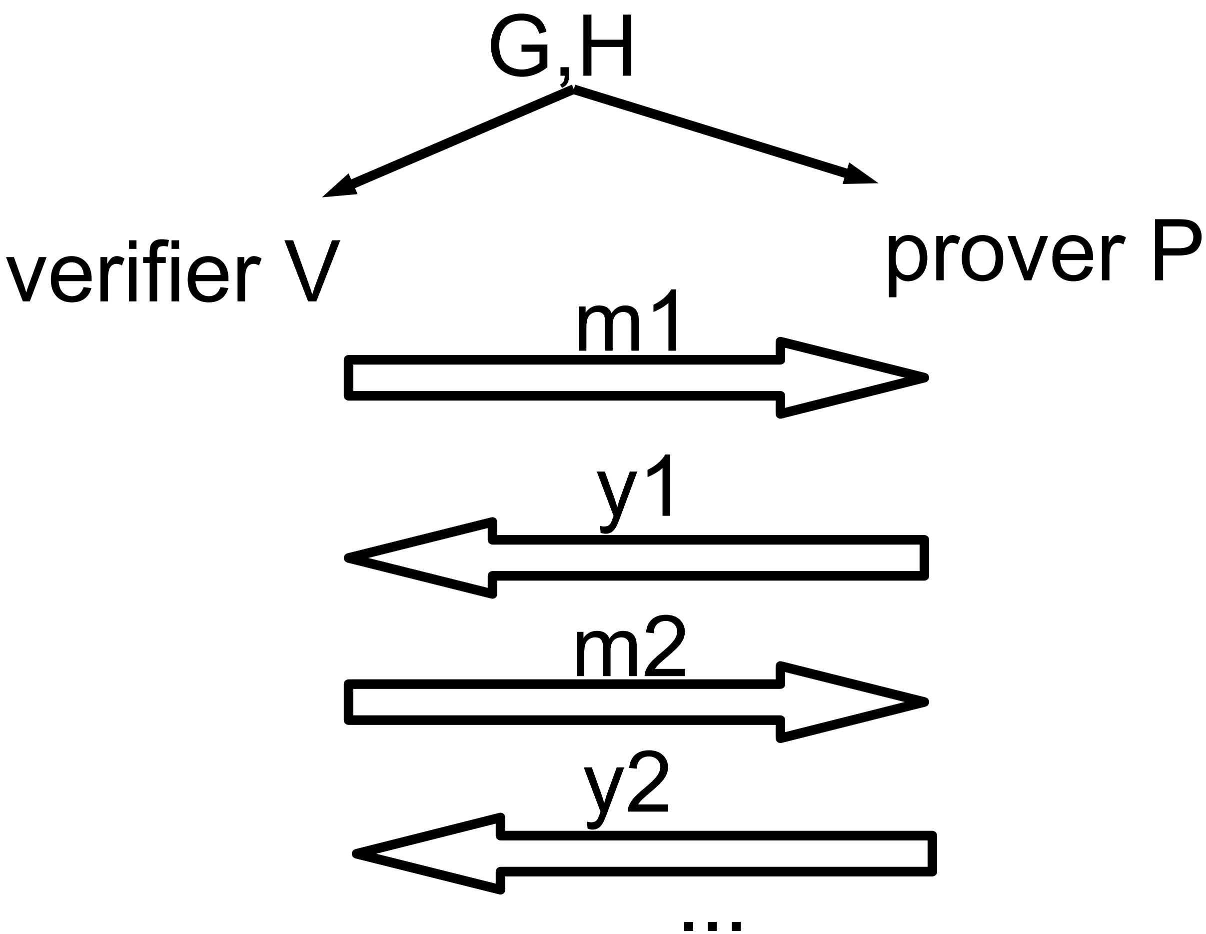
- Open question: 1-message proof system for LABEL-NEQ?



Can a prover send some y that convinces V that G and H are not label-equivalent?

- LABEL-NEQ = {(G,H) | G and H are graphs that are **not** label-equivalent}

- Fact: ∃ interactive proof system for LABEL-NEQ.

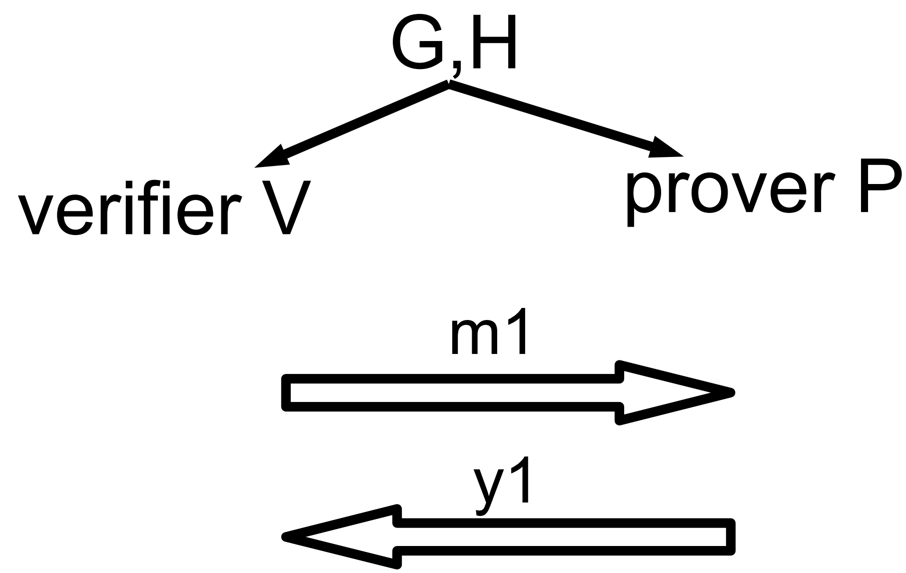


We will now see how this proof system works.

V accepts with high probability ⇔ (G,H) ∈ LABEL-NEQ

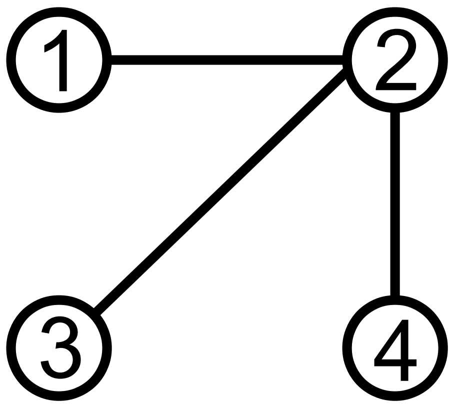
- Fact: ∃ interactive proof system for LABEL-NEQ.

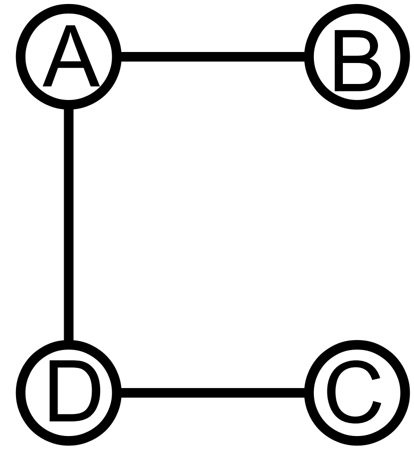
- Proof system:
  - V chooses either G or H, relabels it, sends it to P (m1)
  - P replies "G" or "H" (y1)
  - V accepts ⇔ reply is correct



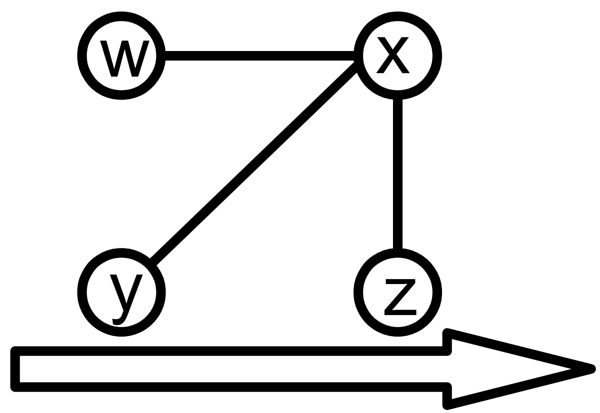
- $(G,H) \in$ LABEL-NEQ $\Rightarrow$ relabeled graph only matches one of G or H: P can answer
- $(G,H) \notin$ LABEL-NEQ $\Rightarrow$ relabeled graph matches both: P is wrong ½ the time

EXAMPLE: (G,H) ∈LABEL-NEQ

G = 

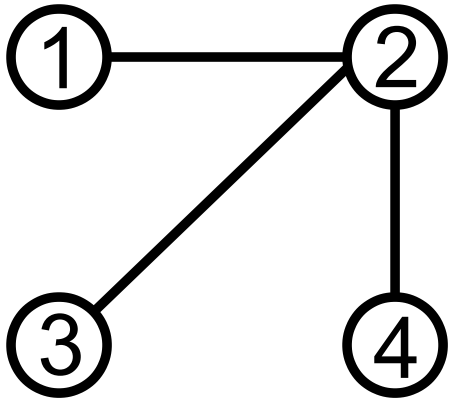
H = 

1) V chooses G, relabels,
sends to P:

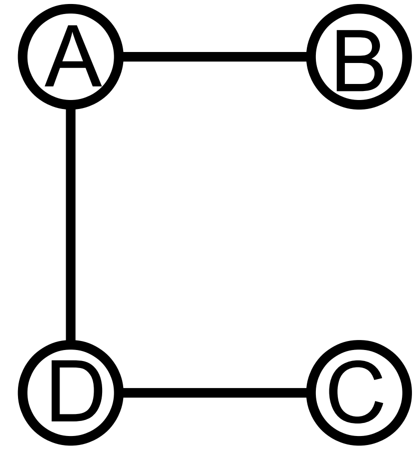


EXAMPLE: (G,H) $\in$ LABEL-NEQ

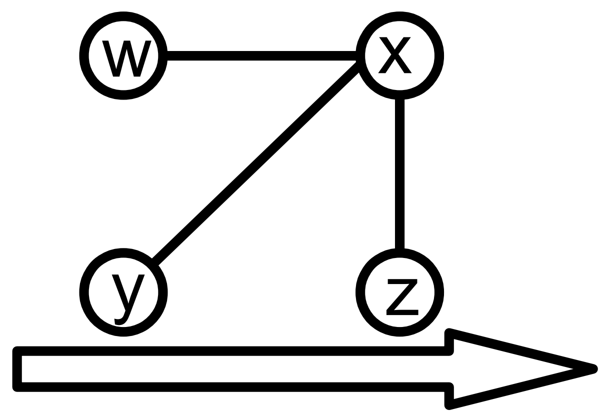
G =



H =



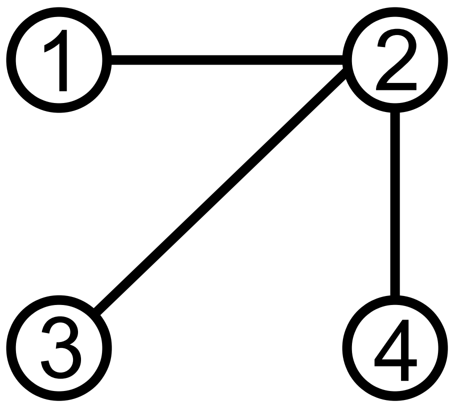
1) V chooses G, relabels,
sends to P:



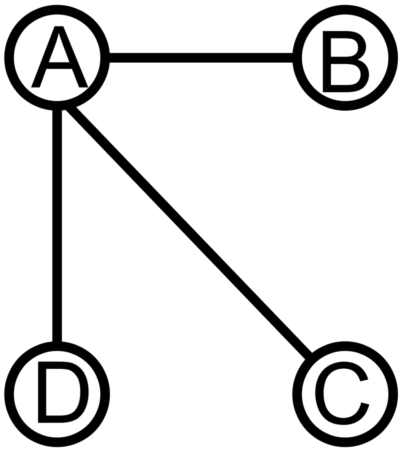
2) P finds mapping (1→w, 2→x, 3→y, 4→z)
and correctly replies: "G"

EXAMPLE: (G,H) $\notin$ LABEL-NEQ

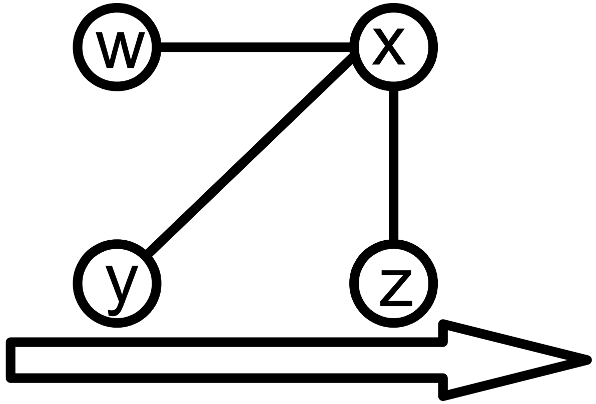
G =



H =

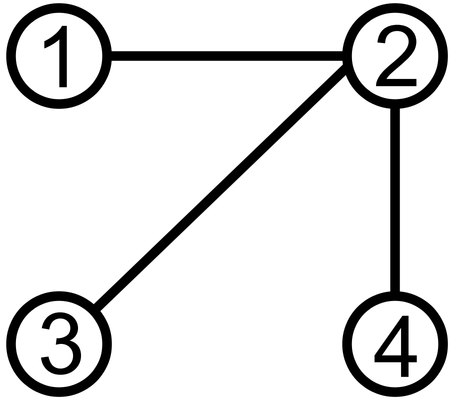


1) V chooses G, relabels,
sends to P:

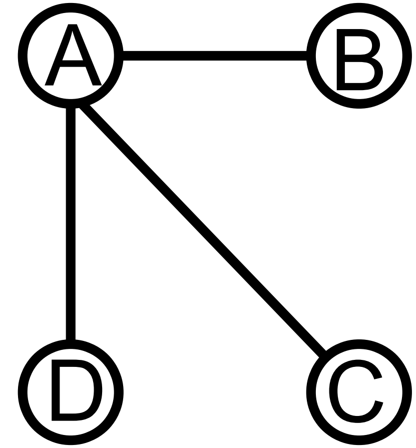


EXAMPLE: (G,H) $\notin$ LABEL-NEQ

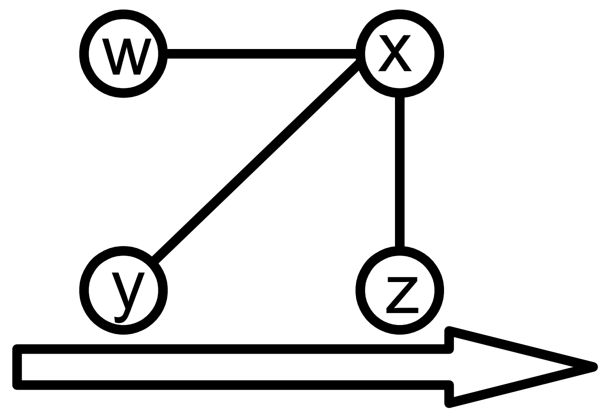
G =



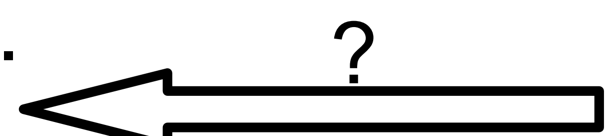
H =



1) V chooses G, relabels, sends to P:



2) P finds two mappings (1→w, 2→x, 3→y, 4→z)
(A→x, B→z, C→y, D→w)
so it doesn't know if V chose G or H.



- Fact: ∃ interactive proof system for LABEL-NEQ.

- (G,H)∈LABEL-NEQ ⇒ relabeled graph only matches one of G or H: P can answer
- (G,H)∉LABEL-NEQ ⇒ relabeled graph matches both: P is wrong ½ the time

Repeat the interaction 100 times:

- (G,H)∈LABEL-NEQ ⇒ P correct every time
- (G,H)∉LABEL-NEQ ⇒ P will be wrong ≥ once (except w/ probability 2^{-100})

V accepts ⇔ P correct every time.

Zero-knowledge proofs

Consider proof system for SAT

Prover's message y reveals more than just the fact that y $\in$ SAT

Is there a proof system which reveals nothing to V, except that the input is in the language?

Such systems are called zero-knowledge

Great achievement: anything in NP has a zero-knowledge proof system

We next show it for 3coloring