

Lecture 23: NP-Completeness I

Jessica Sorrell

November 18, 2025

601.433/633 Introduction to Algorithms

Slides by Michael Dinitz

Introduction

Last few weeks: slower and slower algorithms for harder and harder problems

- ▶ From $O(m + n)$ time algorithms for BFS/DFS/topological sort/SCCs, to $O(m^2n)$ for max flow
- ▶ Today: start of two lectures on NP-completeness.
 - ▶ The (or at least a) line between tractability and intractability

Introduction

Last few weeks: slower and slower algorithms for harder and harder problems

- ▶ From $O(m + n)$ time algorithms for BFS/DFS/topological sort/SCCs, to $O(m^2n)$ for max flow
- ▶ Today: start of two lectures on NP-completeness.
 - ▶ The (or at least a) line between tractability and intractability

Definition

An algorithm runs in *polynomial time* if its (worst-case) running time is $O(n^c)$ for some constant $c \geq 0$, where n is the size of the input.

Think of polynomial time as “fast”, super-polynomial time as “slow”

Introduction

Last few weeks: slower and slower algorithms for harder and harder problems

- ▶ From $O(m + n)$ time algorithms for BFS/DFS/topological sort/SCCs, to $O(m^2n)$ for max flow
- ▶ Today: start of two lectures on NP-completeness.
 - ▶ The (or at least a) line between tractability and intractability

Definition

An algorithm runs in *polynomial time* if its (worst-case) running time is $O(n^c)$ for some constant $c \geq 0$, where n is the size of the input.

Think of polynomial time as “fast”, super-polynomial time as “slow”

Question: When do polynomial-time algorithms exist?

Decision Problems

Definition

A *decision problem* is a computational problem in which the output is either YES or NO.

Decision Problems

Definition

A *decision problem* is a computational problem in which the output is either YES or NO.

Examples:

- ▶ Max-Flow: Input is $G = (V, E), c: E \rightarrow \mathbb{R}_{\geq 0}, s, t \in V, k \in \mathbb{R}^+$. Output YES if there is an (s, t) -flow of value at least k , otherwise output NO.
- ▶ Shortest $s - t$ path: Input is $G = (V, E), \ell: E \rightarrow \mathbb{R}, s, t \in V, k \in \mathbb{R}$. Output YES if $d(s, t) \leq k$, otherwise output NO.

Decision Problems

Definition

A *decision problem* is a computational problem in which the output is either YES or NO.

Examples:

- ▶ Max-Flow: Input is $G = (V, E), c : E \rightarrow \mathbb{R}_{\geq 0}, s, t \in V, k \in \mathbb{R}^+$. Output YES if there is an (s, t) -flow of value at least k , otherwise output NO.
- ▶ Shortest $s - t$ path: Input is $G = (V, E), \ell : E \rightarrow \mathbb{R}, s, t \in V, k \in \mathbb{R}$. Output YES if $d(s, t) \leq k$, otherwise output NO.

Some problems naturally decision, others naturally optimization, but can turn any optimization problem into a decision problem.

- ▶ If can solve decision, can almost always solve optimization.

Decision Problems

Definition

A *decision problem* is a computational problem in which the output is either YES or NO.

Examples:

- ▶ Max-Flow: Input is $G = (V, E), c : E \rightarrow \mathbb{R}_{\geq 0}, s, t \in V, k \in \mathbb{R}^+$. Output YES if there is an (s, t) -flow of value at least k , otherwise output NO.
- ▶ Shortest $s - t$ path: Input is $G = (V, E), \ell : E \rightarrow \mathbb{R}, s, t \in V, k \in \mathbb{R}$. Output YES if $d(s, t) \leq k$, otherwise output NO.

Some problems naturally decision, others naturally optimization, but can turn any optimization problem into a decision problem.

- ▶ If can solve decision, can almost always solve optimization.

Note: Can divide instances (inputs) of any decision problem into YES-instances and NO-instances

Definition

P is the set of decision problems that can be solved in polynomial time.

Note: *problems* are in **P** , not *algorithms*

Definition

P is the set of decision problems that can be solved in polynomial time.

Note: *problems* are in **P** , not *algorithms*

Question: Are all decision problems in **P** ?

Definition

P is the set of decision problems that can be solved in polynomial time.

Note: *problems* are in **P** , not *algorithms*

Question: Are all decision problems in **P** ?

Answer: No!

Definition

P is the set of decision problems that can be solved in polynomial time.

Note: *problems* are in P , not *algorithms*

Question: Are all decision problems in P ?

Answer: No!

- ▶ By *time hierarchy theorem* there are problems that require super-polynomial time!
- ▶ Undecidability: there are problems which cannot be solved by *any* algorithm at all!

Verification

Different Setting: If *in addition* to the input we're given a purported solution, can we check that this solution is valid/feasible (in polynomial time)?

- ▶ Max-Flow: given $\mathbf{f} : \mathbf{E} \rightarrow \mathbb{R}_{\geq 0}$, check that value $\geq k$, flow conservation at all nodes other than \mathbf{s}, \mathbf{t} , and capacity constraints obeyed

Verification

Different Setting: If *in addition* to the input we're given a purported solution, can we check that this solution is valid/feasible (in polynomial time)?

- ▶ Max-Flow: given $f : E \rightarrow \mathbb{R}_{\geq 0}$, check that value $\geq k$, flow conservation at all nodes other than s, t , and capacity constraints obeyed

Definition (3-Coloring)

Input: Undirected graph $G = (V, E)$

Output: YES if \exists coloring $f : V \rightarrow \{R, G, B\}$ such that $f(u) \neq f(v)$ for all $\{u, v\} \in E$. NO otherwise

Verification

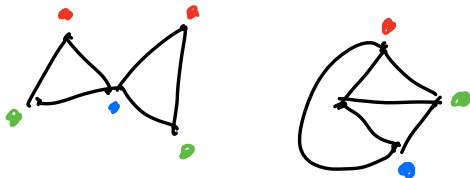
Different Setting: If *in addition* to the input we're given a purported solution, can we check that this solution is valid/feasible (in polynomial time)?

- ▶ Max-Flow: given $f : E \rightarrow \mathbb{R}_{\geq 0}$, check that value $\geq k$, flow conservation at all nodes other than s, t , and capacity constraints obeyed

Definition (3-Coloring)

Input: Undirected graph $G = (V, E)$

Output: YES if \exists coloring $f : V \rightarrow \{R, G, B\}$ such that $f(u) \neq f(v)$ for all $\{u, v\} \in E$. NO otherwise



Verification

Different Setting: If *in addition* to the input we're given a purported solution, can we check that this solution is valid/feasible (in polynomial time)?

- ▶ Max-Flow: given $f : E \rightarrow \mathbb{R}_{\geq 0}$, check that value $\geq k$, flow conservation at all nodes other than s, t , and capacity constraints obeyed

Definition (3-Coloring)

Input: Undirected graph $G = (V, E)$

Output: YES if \exists coloring $f : V \rightarrow \{R, G, B\}$ such that $f(u) \neq f(v)$ for all $\{u, v\} \in E$. NO otherwise

Verification: Given f ,

- ▶ Check that $f(u) \in \{R, G, B\}$ for all $u \in V$, and
- ▶ Check each edge $\{u, v\}$ to make sure that $f(u) \neq f(v)$

NP

NP: decision problems where solutions can be *verified* in polynomial time.

Definition

A decision problem Q is in **NP** (*nondeterministic polynomial time*) if there exists a polynomial time algorithm $V(I, X)$ (called the *verifier*) such that

1. If I is a YES-instance of Q , then there is some X (usually called the *witness*, *proof*, or *solution*) with size polynomial in $|I|$ so that $V(I, X) = \text{YES}$.
2. If I is a NO-instance of Q , then $V(I, X) = \text{NO}$ for all X .

NP

NP: decision problems where solutions can be *verified* in polynomial time.

Definition

A decision problem Q is in **NP** (*nondeterministic polynomial time*) if there exists a polynomial time algorithm $V(I, X)$ (called the *verifier*) such that

1. If I is a YES-instance of Q , then there is some X (usually called the *witness*, *proof*, or *solution*) with size polynomial in $|I|$ so that $V(I, X) = \text{YES}$.
2. If I is a NO-instance of Q , then $V(I, X) = \text{NO}$ for all X .

Examples:

- ▶ 3-coloring: Witness X is a coloring $f : V \rightarrow \{R, B, G\}$, verifier checks each edge $\{u, v\}$ to make sure $f(u) \neq f(v)$
 - ▶ If I is a YES instance, then there is a coloring so verifier will return YES
 - ▶ If I is a NO instance, then no valid coloring exists. Whatever X is, verifier returns NO.

NP

NP: decision problems where solutions can be *verified* in polynomial time.

Definition

A decision problem Q is in **NP** (*nondeterministic polynomial time*) if there exists a polynomial time algorithm $V(I, X)$ (called the *verifier*) such that

1. If I is a YES-instance of Q , then there is some X (usually called the *witness*, *proof*, or *solution*) with size polynomial in $|I|$ so that $V(I, X) = \text{YES}$.
2. If I is a NO-instance of Q , then $V(I, X) = \text{NO}$ for all X .

Examples:

- ▶ Max-Flow: Witness X is a flow $f : E \rightarrow \mathbb{R}_{\geq 0}$, verifier checks that it's feasible of value $\geq k$
 - ▶ If I is a YES instance, then there is a feasible flow of value at least k so verifier (on this flow) will return YES
 - ▶ If I a NO instance, then no feasible flow of value $\geq k$. Whatever X is, verifier returns NO.

NP

NP: decision problems where solutions can be *verified* in polynomial time.

Definition

A decision problem Q is in **NP** (*nondeterministic polynomial time*) if there exists a polynomial time algorithm $V(I, X)$ (called the *verifier*) such that

1. If I is a YES-instance of Q , then there is some X (usually called the *witness*, *proof*, or *solution*) with size polynomial in $|I|$ so that $V(I, X) = \text{YES}$.
2. If I is a NO-instance of Q , then $V(I, X) = \text{NO}$ for all X .

Examples:

- ▶ Factoring: Instance is pair of integers M, k . YES if M has as factor in $\{2, \dots, k\}$, NO otherwise.
 - ▶ Witness: integer f in $\{2, 3, \dots, k\}$. Verifier: returns YES if M/f is an integer and $f \in \{2, \dots, k\}$, NO otherwise.
 - ▶ If YES instance, then an f does exist so verifier returns YES on that f . If NO, then no such f exists so verifier always returns NO.

NP

NP: decision problems where solutions can be *verified* in polynomial time.

Definition

A decision problem Q is in **NP** (*nondeterministic polynomial time*) if there exists a polynomial time algorithm $V(I, X)$ (called the *verifier*) such that

1. If I is a YES-instance of Q , then there is some X (usually called the *witness*, *proof*, or *solution*) with size polynomial in $|I|$ so that $V(I, X) = \text{YES}$.
2. If I is a NO-instance of Q , then $V(I, X) = \text{NO}$ for all X .

Examples:

- ▶ Traveling Salesman: Instance is weighted graph G and integer k . YES iff G has a tour (walk that touches every vertex at least once) of length $\leq k$.
 - ▶ Witness: tour P . Verifier checks that it is a tour, has length at most k
 - ▶ If YES instance, then such a tour exists \implies verifier returns YES on that tour.
 - ▶ If NO, no such tour exists \implies verifier always returns NO.

NP

NP: decision problems where solutions can be *verified* in polynomial time.

Definition

A decision problem Q is in **NP** (*nondeterministic polynomial time*) if there exists a polynomial time algorithm $V(I, X)$ (called the *verifier*) such that

1. If I is a YES-instance of Q , then there is some X (usually called the *witness*, *proof*, or *solution*) with size polynomial in $|I|$ so that $V(I, X) = \text{YES}$.
2. If I is a NO-instance of Q , then $V(I, X) = \text{NO}$ for all X .

Important asymmetry: need a witness for YES, not a witness for NO.

P vs NP

Theorem

$$P \subseteq NP$$

P vs NP

Theorem

$$P \subseteq NP$$

Proof.

Let $Q \in P$.

$V(I, X)$: Ignore X , solve on instance I .



P vs NP

Theorem

$$P \subseteq NP$$

Proof.

Let $Q \in P$.

$V(I, X)$: Ignore X , solve on instance I .



Question: Does $P = NP$, i.e., is $NP \subseteq P$?



P vs NP

Theorem

$$P \subseteq NP$$

Proof.

Let $Q \in P$.

$V(I, X)$: Ignore X , solve on instance I .



Question: Does $P = NP$, i.e., is $NP \subseteq P$?

- ▶ *Almost* everyone thinks no, but we don't know for sure!
- ▶ Not even particularly close to a proof.
- ▶ Think about what $P = NP$ would mean...



Reductions

Question: How could we prove that $P = NP$ or $P \neq NP$?

Reductions

Question: How could we prove that $P = NP$ or $P \neq NP$?

- ▶ $P = NP$: Need to show that *every* problem in NP is also in P !
- ▶ $P \neq NP$: Need to prove that *some* problem in NP not in P .
 - ▶ What is the “hardest” problem in NP ?

Reductions

Question: How could we prove that $P = NP$ or $P \neq NP$?

- ▶ $P = NP$: Need to show that *every* problem in NP is also in P !
- ▶ $P \neq NP$: Need to prove that *some* problem in NP not in P .
 - ▶ What is the “hardest” problem in NP ?

Definition

Problem A is *polynomial-time reducible* to problem B (written $A \leq_p B$) if, given a polynomial-time algorithm for B , we can use it to produce a polynomial-time algorithm for A .

Reductions

Question: How could we prove that $P = NP$ or $P \neq NP$?

- ▶ $P = NP$: Need to show that *every* problem in NP is also in P !
- ▶ $P \neq NP$: Need to prove that *some* problem in NP not in P .
 - ▶ What is the “hardest” problem in NP ?

Definition

Problem A is *polytime reducible* to problem B (written $A \leq_p B$) if, given a polynomial-time algorithm for B , we can use it to produce a polynomial-time algorithm for A .

Means that B is “at least as hard” as A : if B is in P , then so is A .

- ▶ So “hardest” problems in NP are problems that many other problems reduce to.

Many-One (Karp) Reductions

Almost always (and always in this course), use a special type of reduction.

Definition

A *Many-one* or *Karp* reduction from \mathbf{A} to \mathbf{B} is a function f which takes arbitrary instances of \mathbf{A} and transforms them into instances of \mathbf{B} so that

1. If x is a YES-instance of \mathbf{A} then $f(x)$ is a YES-instance of \mathbf{B} .
2. If x is a NO-instance of \mathbf{A} then $f(x)$ is a NO-instance \mathbf{B} .
3. f can be computed in polynomial time.

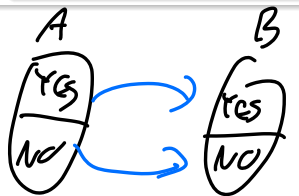
Many-One (Karp) Reductions

Almost always (and always in this course), use a special type of reduction.

Definition

A *Many-one* or *Karp* reduction from A to B is a function f which takes arbitrary instances of A and transforms them into instances of B so that

1. If x is a YES-instance of A then $f(x)$ is a YES-instance of B .
2. If x is a NO-instance of A then $f(x)$ is a NO-instance of B .
3. f can be computed in polynomial time.



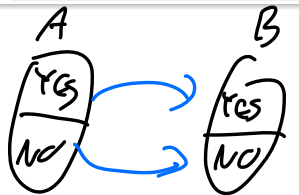
Many-One (Karp) Reductions

Almost always (and always in this course), use a special type of reduction.

Definition

A *Many-one* or *Karp* reduction from A to B is a function f which takes arbitrary instances of A and transforms them into instances of B so that

1. If x is a YES-instance of A then $f(x)$ is a YES-instance of B .
2. If x is a NO-instance of A then $f(x)$ is a NO-instance of B .
3. f can be computed in polynomial time.



So given instance x of A , compute $f(x)$ and use polytime algorithm for B on $f(x)$

- ▶ Polytime, since f in polytime and algorithm for B in polytime
- ▶ Correct by first two properties of many-one reduction.

NP-Completeness

So what is “hardest problem” in **NP**?

Definition

Problem **Q** is **NP-hard** if $Q' \leq_p Q$ for all problems Q' in **NP**.

Definition

Problem **Q** is **NP-complete** if it is **NP-hard** and in **NP**.

NP-Completeness

So what is “hardest problem” in **NP**?

Definition

Problem **Q** is **NP-hard** if $Q' \leq_p Q$ for all problems Q' in **NP**.

Definition

Problem **Q** is **NP-complete** if it is **NP-hard** and in **NP**.

So suppose **Q** is **NP-complete**.

- ▶ To prove $P \neq NP$: Hardest problem in **NP**! If anything in **NP** is not in **P**, then **Q** is not in **P**
- ▶ To prove $P = NP$: Just need to prove that $Q \in P$.

NP-Completeness

So what is “hardest problem” in **NP**?

Definition

Problem **Q** is **NP-hard** if $Q' \leq_p Q$ for all problems Q' in **NP**.

Definition

Problem **Q** is **NP-complete** if it is **NP-hard** and in **NP**.

So suppose **Q** is **NP-complete**.

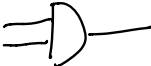
- ▶ To prove $P \neq NP$: Hardest problem in **NP**! If anything in **NP** is not in **P**, then **Q** is not in **P**
- ▶ To prove $P = NP$: Just need to prove that $Q \in P$.

Is anything **NP-complete**?


Circuit-SAT

Definition

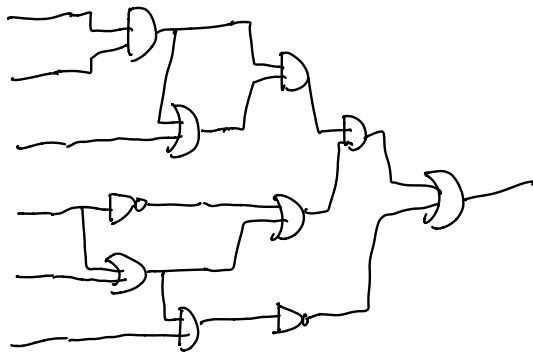
Circuit-SAT: Given a boolean circuit with a single output and no loops (some inputs might be hardwired), is there a way of setting the inputs so that the output of the circuit is **1**?

Gates: AND 

OR 

NOT 

Arbitrary fan-out



Circuit-SAT

Theorem

*Circuit-SAT is **NP**-complete.*

Sketch of proof here. See book for details.

Circuit-SAT

Theorem

*Circuit-SAT is **NP**-complete.*

Sketch of proof here. See book for details.

Lemma

*Circuit-SAT is in **NP**.*

Circuit-SAT

Theorem

*Circuit-SAT is **NP**-complete.*

Sketch of proof here. See book for details.

Lemma

*Circuit-SAT is in **NP**.*

Proof.

Witness is a T/F (or 1/0) assignment to inputs. Verifier simulates circuit on assignment, checks that it outputs **1**.

Circuit-SAT

Theorem

*Circuit-SAT is **NP**-complete.*

Sketch of proof here. See book for details.

Lemma

*Circuit-SAT is in **NP**.*

Proof.

Witness is a T/F (or 1/0) assignment to inputs. Verifier simulates circuit on assignment, checks that it outputs **1**.

- ▶ If input is a YES instance then there is some assignment so circuit outputs **1**. When verifier run on that assignment, returns YES.
- ▶ In input is a NO instance then in every assignment circuit outputs **0**. So verifier returns NO on every witness.



Circuit-SAT is *NP*-hard

Let $A \in NP$. Want to show $A \leq_p \text{Circuit-SAT}$ (construct a many-one reduction).

Circuit-SAT is *NP*-hard

Let $A \in NP$. Want to show $A \leq_p \text{Circuit-SAT}$ (construct a many-one reduction).

Where to start? What do we know about A ?

Circuit-SAT is *NP*-hard

Let $A \in NP$. Want to show $A \leq_p \text{Circuit-SAT}$ (construct a many-one reduction).

Where to start? What do we know about A ?

- ▶ In *NP*, so has verifier algorithm V
- ▶ V algorithm runs on a computer (or Turing machine)!

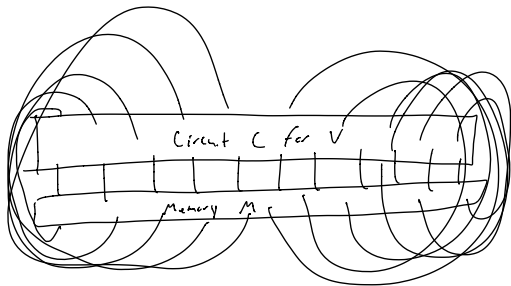
Circuit-SAT is *NP*-hard

Let $A \in NP$. Want to show $A \leq_p$ Circuit-SAT (construct a many-one reduction).

Where to start? What do we know about A ?

- ▶ In *NP*, so has verifier algorithm V
- ▶ V algorithm runs on a computer (or Turing machine)!

Computer: memory + circuit for modifying memory!



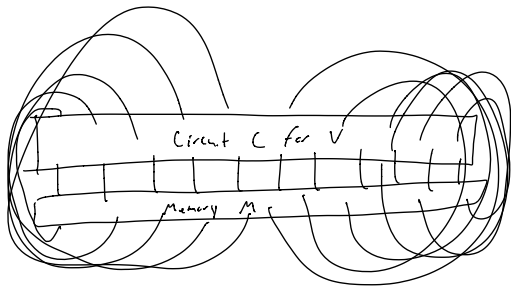
Circuit-SAT is *NP*-hard

Let $A \in NP$. Want to show $A \leq_p$ Circuit-SAT (construct a many-one reduction).

Where to start? What do we know about A ?

- ▶ In *NP*, so has verifier algorithm V
- ▶ V algorithm runs on a computer (or Turing machine)!

Computer: memory + circuit for modifying memory!



Not a boolean circuit in Circuit-SAT sense: loops (feedback)

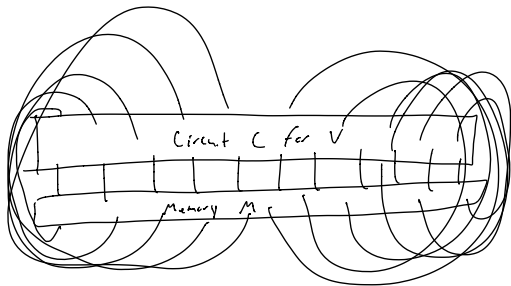
Circuit-SAT is *NP*-hard

Let $A \in NP$. Want to show $A \leq_p$ Circuit-SAT (construct a many-one reduction).

Where to start? What do we know about A ?

- ▶ In *NP*, so has verifier algorithm V
- ▶ V algorithm runs on a computer (or Turing machine)!

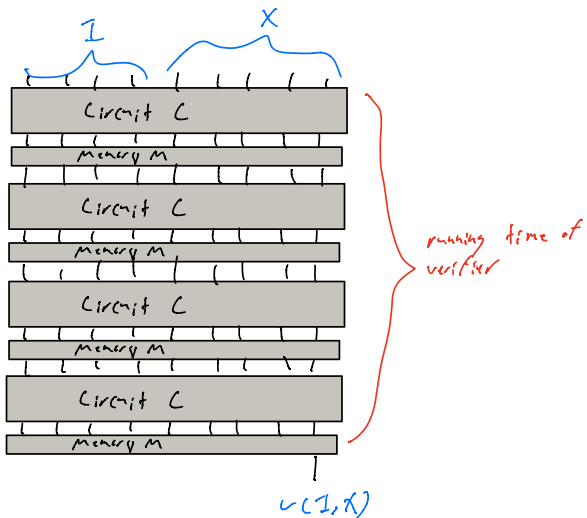
Computer: memory + circuit for modifying memory!



Not a boolean circuit in Circuit-SAT sense: loops (feedback)

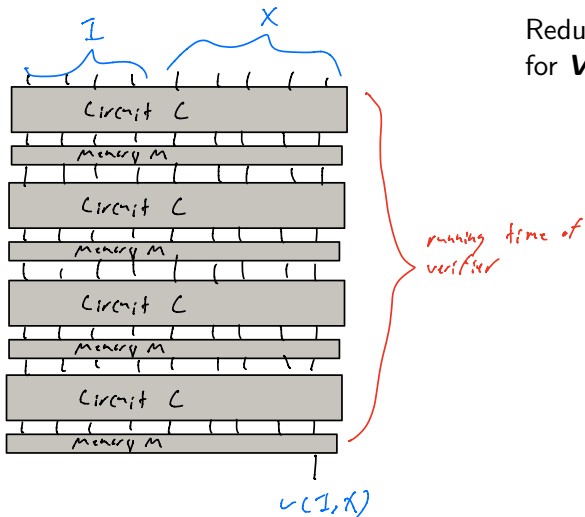
Fix: “Unroll” circuit using fact that V runs in polynomial time

Reduction

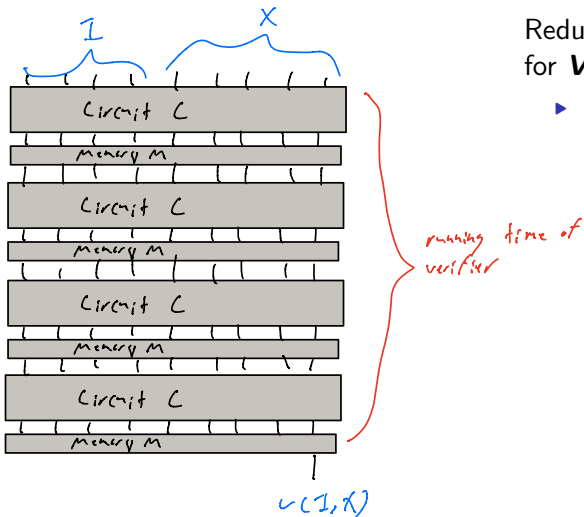


Reduction

Reduction: given instance I of A , construct this circuit for V , hardwire I . Combined circuit $f(I)$



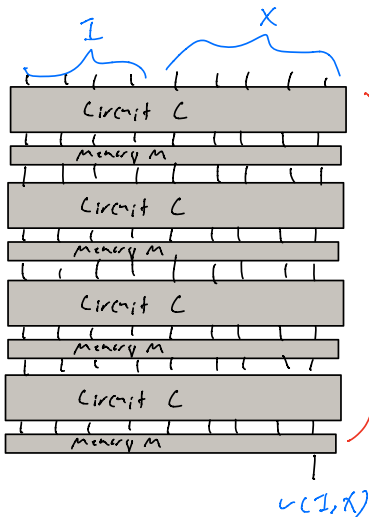
Reduction



Reduction: given instance I of A , construct this circuit for V , hardwire I . Combined circuit $f(I)$

- Polytime since V runs in polytime

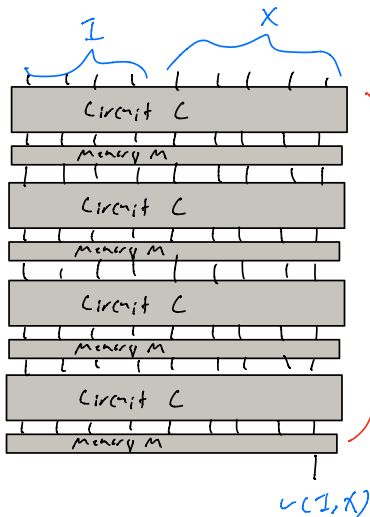
Reduction



Reduction: given instance I of A , construct this circuit for V , hardwire I . Combined circuit $f(I)$

- ▶ Polytime since V runs in polytime
- ▶ If I YES of A : there is some X so that $V(I, X) = \text{YES}$
 \implies some X so that when X input to $f(I)$, outputs 1
 $\implies f(I)$ YES instance of Circuit-SAT.

Reduction



Reduction: given instance I of A , construct this circuit for V , hardwire I . Combined circuit $f(I)$

- ▶ Polytime since V runs in polytime
- ▶ If I YES of A : there is some X so that $V(I, X) = \text{YES}$
 \implies some X so that when X input to $f(I)$, outputs 1
 $\implies f(I)$ YES instance of Circuit-SAT.
- ▶ If I NO of A : For every X , know that $V(I, X) = \text{NO}$
 \implies for every X , when X input to $f(I)$, outputs 0
 $\implies f(I)$ NO instance of Circuit-SAT