NP-Completeness

Peter Lammich

March 2, 2020

Last Lecture

- Decision Problems: Language = accepted bit-strings

Last Lecture

- Decision Problems: Language = accepted bit-strings
- Reduction: $A \leq_p B$. Encode problem A as problem B
  - $x \in L_A \iff f(x) \in L_B$
  - f computable in polynomial time
  - A not harder than B

Last Lecture

- Decision Problems: Language = accepted bit-strings
- Reduction: $A \leq_p B$. Encode problem A as problem B
  - $x \in L_A \iff f(x) \in L_B$
  - f computable in polynomial time
  - A not harder than B
- Certificate: extra information to verify membership in language
  - Sound and complete: $(\exists c.\ check(w, c)) \iff w \in L$
  - Example: solution of Boolean formula

Last Lecture

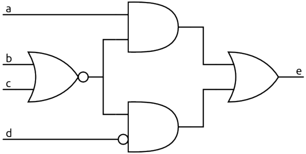
- Decision Problems: Language = accepted bit-strings
- Reduction: $A \leq_p B$. Encode problem A as problem B
  - $x \in L_A \iff f(x) \in L_B$
  - f computable in polynomial time
  - A not harder than B
- Certificate: extra information to verify membership in language
  - Sound and complete: $(\exists c.\ check(w, c)) \iff w \in L$
  - Example: solution of Boolean formula
- NP: problems with poly-time certificates
  - NP-hard: harder than any problem in NP
  - NP-complete = in NP + NP-hard

Last Lecture

- Decision Problems: Language = accepted bit-strings
- Reduction: $A \leq_p B$. Encode problem A as problem B
  - $x \in L_A \iff f(x) \in L_B$
  - f computable in polynomial time
  - A not harder than B
- Certificate: extra information to verify membership in language
  - Sound and complete: $(\exists c.\ check(w, c)) \iff w \in L$
  - Example: solution of Boolean formula
- NP: problems with poly-time certificates
  - NP-hard: harder than any problem in NP
  - NP-complete = in NP + NP-hard
- How to show that problem is ...
  - ... in NP: show it has poly-time certificates
  - ... NP-hard: reduce another NP-hard problem to it

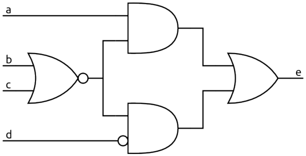
Circuit-SAT

- Given a combinational circuit with n gates, $m \leq 2n$ inputs, and one output
  - Let's restrict gate types to AND, OR, NOT



Circuit-SAT

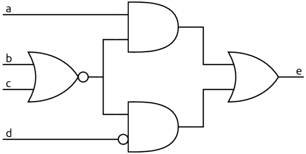
- Given a combinational circuit with n gates, $m \leq 2n$ inputs, and one output
  - Let's restrict gate types to AND, OR, NOT



- Can the inputs be driven such that the output goes to 1?

Circuit-SAT

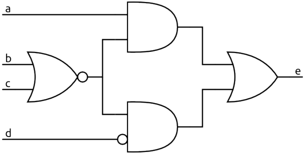
- Given a combinational circuit with n gates, $m \leq 2n$ inputs, and one output
  - Let's restrict gate types to AND, OR, NOT



- Can the inputs be driven such that the output goes to 1?
- Circuit-SAT is in NP
  - given the inputs, simulate the circuit and check the output
  - obviously polynomial time!

Circuit-SAT

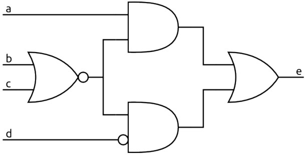
- Given a combinational circuit with n gates, $m \leq 2n$ inputs, and one output
  - Let's restrict gate types to AND, OR, NOT



- Can the inputs be driven such that the output goes to 1?
- Circuit-SAT is in NP
  - given the inputs, simulate the circuit and check the output
  - obviously polynomial time!
- Now: Circuit-SAT is NP-hard

Circuit-SAT

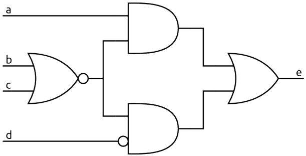
- Given a combinational circuit with n gates, $m \leq 2n$ inputs, and one output
  - Let's restrict gate types to AND, OR, NOT



- Can the inputs be driven such that the output goes to 1?
- Circuit-SAT is in NP
  - given the inputs, simulate the circuit and check the output
  - obviously polynomial time!
- Now: Circuit-SAT is NP-hard
  - only sketch of proof

Circuit-SAT

- Given a combinational circuit with n gates, $m \leq 2n$ inputs, and one output
  - Let's restrict gate types to AND, OR, NOT



- Can the inputs be driven such that the output goes to 1?
- Circuit-SAT is in NP
  - given the inputs, simulate the circuit and check the output
  - obviously polynomial time!
- Now: Circuit-SAT is NP-hard
  - only sketch of proof
  - precise proof: lot's of subtle technical details

Proof Sketch: Circuit-SAT is NP-hard

Reduce some arbitrary problem $A \in NP$ to Circuit-SAT

Proof Sketch: Circuit-SAT is NP-hard

Reduce some arbitrary problem $A \in NP$ to Circuit-SAT

- A has poly-time certificates.

Proof Sketch: Circuit-SAT is NP-hard

Reduce some arbitrary problem $A \in NP$ to Circuit-SAT

- A has poly-time certificates.
    - $check(w, c)$ function that runs in time $< (|w| + |c|)^{k_1}$

Proof Sketch: Circuit-SAT is NP-hard

Reduce some arbitrary problem $A \in NP$ to Circuit-SAT

- A has poly-time certificates.
  - $check(w, c)$ function that runs in time $< (|w| + |c|)^{k_1}$
  - any word $w \in L$ has certificate c, with $|c| < |w|^{k_2}$

Proof Sketch: Circuit-SAT is NP-hard

Reduce some arbitrary problem $A \in NP$ to Circuit-SAT

- A has poly-time certificates.
  - $check(w, c)$ function that runs in time $< (|w| + |c|)^{k_1}$
  - any word $w \in L$ has certificate c, with $|c| < |w|^{k_2}$
- To check $w \in L_A$, we construct a circuit

Proof Sketch: Circuit-SAT is NP-hard

Reduce some arbitrary problem $A \in NP$ to Circuit-SAT

- A has poly-time certificates.
  - $check(w, c)$ function that runs in time $< (|w| + |c|)^{k_1}$
  - any word $w \in L$ has certificate c, with $|c| < |w|^{k_2}$
- To check $w \in L_A$, we construct a circuit
  - with $|w|^{k_2}$ inputs and one output

Proof Sketch: Circuit-SAT is NP-hard

Reduce some arbitrary problem $A \in NP$ to Circuit-SAT

- A has poly-time certificates.
  - $check(w, c)$ function that runs in time $< (|w| + |c|)^{k_1}$
  - any word $w \in L$ has certificate c, with $|c| < |w|^{k_2}$
- To check $w \in L_A$, we construct a circuit
  - with $|w|^{k_2}$ inputs and one output
  - that simulates $(|w| + |w|^{k_2})^{k_1}$ steps of $check(w, \cdot)$

Proof Sketch: Circuit-SAT is NP-hard

Reduce some arbitrary problem $A \in NP$ to Circuit-SAT

- A has poly-time certificates.
  - $check(w, c)$ function that runs in time $< (|w| + |c|)^{k_1}$
  - any word $w \in L$ has certificate c, with $|c| < |w|^{k_2}$
- To check $w \in L_A$, we construct a circuit
  - with $|w|^{k_2}$ inputs and one output
  - that simulates $(|w| + |w|^{k_2})^{k_1}$ steps of $check(w, \cdot)$
- Circuit-SAT checks if there are inputs that produce output 1

Proof Sketch: Circuit-SAT is NP-hard

Reduce some arbitrary problem $A \in NP$ to Circuit-SAT

- A has poly-time certificates.
  - $check(w, c)$ function that runs in time $< (|w| + |c|)^{k_1}$
  - any word $w \in L$ has certificate c, with $|c| < |w|^{k_2}$
- To check $w \in L_A$, we construct a circuit
  - with $|w|^{k_2}$ inputs and one output
  - that simulates $(|w| + |w|^{k_2})^{k_1}$ steps of $check(w, \cdot)$
- Circuit-SAT checks if there are inputs that produce output 1
  - YES: the inputs correspond to valid certificate $\implies w \in L_A$

Proof Sketch: Circuit-SAT is NP-hard

Reduce some arbitrary problem $A \in NP$ to Circuit-SAT

- A has poly-time certificates.
  - $check(w, c)$ function that runs in time $< (|w| + |c|)^{k_1}$
  - any word $w \in L$ has certificate c, with $|c| < |w|^{k_2}$
- To check $w \in L_A$, we construct a circuit
  - with $|w|^{k_2}$ inputs and one output
  - that simulates $(|w| + |w|^{k_2})^{k_1}$ steps of $check(w, \cdot)$
- Circuit-SAT checks if there are inputs that produce output 1
  - YES: the inputs correspond to valid certificate $\implies w \in L_A$
  - NO: there is no valid certificate (within size bound) $\implies w \notin L_A$

Proof Sketch: Circuit-SAT is NP-hard

Reduce some arbitrary problem $A \in NP$ to Circuit-SAT

- A has poly-time certificates.
  - $check(w, c)$ function that runs in time $< (|w| + |c|)^{k_1}$
  - any word $w \in L$ has certificate c, with $|c| < |w|^{k_2}$
- To check $w \in L_A$, we construct a circuit
  - with $|w|^{k_2}$ inputs and one output
  - that simulates $(|w| + |w|^{k_2})^{k_1}$ steps of $check(w, \cdot)$
- Circuit-SAT checks if there are inputs that produce output 1
  - YES: the inputs correspond to valid certificate $\implies w \in L_A$
  - NO: there is no valid certificate (within size bound) $\implies w \notin L_A$
- Now: How to construct such a circuit

Let's Build a (simplistic) Computer!

- Ingredients:
  - n bits of memory
  - a control circuit C with n inputs, n outputs, and $poly(n)$ gates

Let's Build a (simplistic) Computer!

- Ingredients:
  - n bits of memory
  - a control circuit C with n inputs, n outputs, and $poly(n)$ gates
- Working mode: in each cycle:
  - feed the memory content to the inputs
  - and then update the memory with the outputs

Let's Build a (simplistic) Computer!

- Ingredients:
  - n bits of memory
  - a control circuit C with n inputs, n outputs, and $poly(n)$ gates
- Working mode: in each cycle:
  - feed the memory content to the inputs
  - and then update the memory with the outputs
- For n bits of memory, a circuit of size $poly(n)$ is enough!
  - and can be constructed in $poly(n)$ time

Let's Build a (simplistic) Computer!

- Ingredients:
  - n bits of memory
  - a control circuit C with n inputs, n outputs, and $poly(n)$ gates
- Working mode: in each cycle:
  - feed the memory content to the inputs
  - and then update the memory with the outputs
- For n bits of memory, a circuit of size $poly(n)$ is enough!
  - and can be constructed in $poly(n)$ time
- How realistic is that?

Let's Build a (simplistic) Computer!

- Ingredients:
  - n bits of memory
  - a control circuit C with n inputs, n outputs, and $poly(n)$ gates
- Working mode: in each cycle:
  - feed the memory content to the inputs
  - and then update the memory with the outputs
- For n bits of memory, a circuit of size $poly(n)$ is enough!
  - and can be constructed in $poly(n)$ time
- How realistic is that?
  - not very realistic! BUT

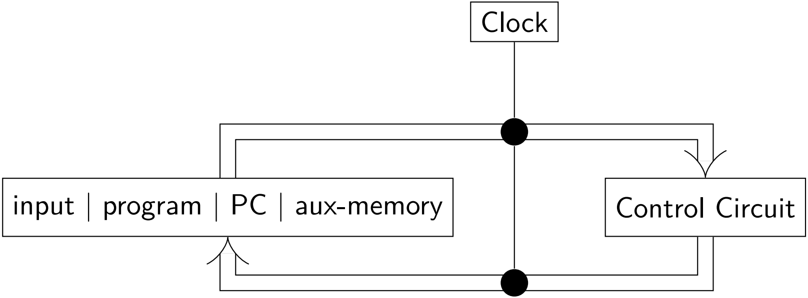
Let's Build a (simplistic) Computer!

- Ingredients:
  - n bits of memory
  - a control circuit C with n inputs, n outputs, and $poly(n)$ gates
- Working mode: in each cycle:
  - feed the memory content to the inputs
  - and then update the memory with the outputs
- For n bits of memory, a circuit of size $poly(n)$ is enough!
  - and can be constructed in $poly(n)$ time
- How realistic is that?
  - not very realistic! BUT
  - powerful enough to run programs

Let's Build a (simplistic) Computer!

- Ingredients:
  - n bits of memory
  - a control circuit C with n inputs, n outputs, and $poly(n)$ gates
- Working mode: in each cycle:
  - feed the memory content to the inputs
  - and then update the memory with the outputs
- For n bits of memory, a circuit of size $poly(n)$ is enough!
  - and can be constructed in $poly(n)$ time
- How realistic is that?
  - not very realistic! BUT
  - powerful enough to run programs
  - easy enough to be simulated by standard computer
    - in $poly(n)$ time per cycle!

Our Computer



Simulating n cycles

- link together $n-1$ copies of the control circuit
  - yields circuit with $(n-1)poly(n) = poly(n)$ gates
  - inputs of first copy: initial memory
  - outputs of last copy: memory after n cycles

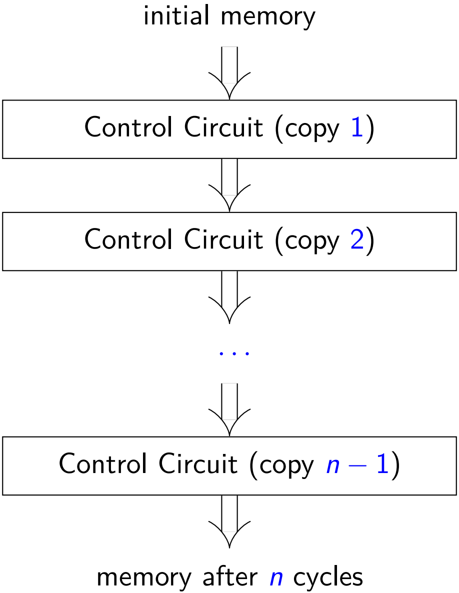
Simulating n cycles

- link together $n - 1$ copies of the control circuit
  - yields circuit with $(n-1)poly(n) = poly(n)$ gates
  - inputs of first copy: initial memory
  - outputs of last copy: memory after n cycles
- initializing some of the inputs
  - partially evaluate the circuit
  - only reduces number of gates
  - here: initialize program, PC, aux-memory.

Simulating n cycles

- link together $n-1$ copies of the control circuit
  - yields circuit with $(n-1)poly(n) = poly(n)$ gates
  - inputs of first copy: initial memory
  - outputs of last copy: memory after n cycles
- initializing some of the inputs
  - partially evaluate the circuit
  - only reduces number of gates
  - here: initialize program, PC, aux-memory.
- ignoring some outputs
  - only reduces number of gates
  - here: keep only the single bit that contains result

Circuit for n cycles



Outlook

- We have an initial NP-complete problem
- We now reduce it to other problems, to show that they are NP-complete, too!
- Circuit-SAT $\leq_p$ SAT $\leq_p$ 3SAT $\leq_p$ CLIQUE $\ldots$

SAT

- Is a Boolean formula with m operators and n variables satisfiable?

$$\exists x_1 \ldots x_n.\ x_1 \wedge (x_2 \vee \neg x_3) \vee x_4 \ldots$$

SAT

- Is a Boolean formula with m operators and n variables satisfiable?

$$\exists x_1 \ldots x_n.\ x_1 \wedge (x_2 \vee \neg x_3) \vee x_4 \ldots$$

- SAT is in NP: solution can be checked by evaluating formula.
  - obviously in polynomial time!

SAT

- Is a Boolean formula with m operators and n variables satisfiable?

$$\exists x_1 \ldots x_n.\ x_1 \wedge (x_2 \vee \neg x_3) \vee x_4 \ldots$$

- SAT is in NP: solution can be checked by evaluating formula.
  - obviously in polynomial time!
- SAT is NP-hard: show Circuit-SAT $\leq_p$ SAT

SAT

- Is a Boolean formula with m operators and n variables satisfiable?

$$\exists x_1 \ldots x_n .\ x_1 \wedge (x_2 \vee \neg x_3) \vee x_4 \ldots$$

- SAT is in NP: solution can be checked by evaluating formula.
  - obviously in polynomial time!
- SAT is NP-hard: show Circuit-SAT $\leq_p$ SAT
  - Introduce one variable per wire (input, output, internal)

SAT

- Is a Boolean formula with m operators and n variables satisfiable?

$$\exists x_1 \dots x_n.\ x_1 \wedge (x_2 \vee \neg x_3) \vee x_4 \dots$$

- SAT is in NP: solution can be checked by evaluating formula.
  - obviously in polynomial time!
- SAT is NP-hard: show Circuit-SAT $\leq_p$ SAT
  - Introduce one variable per wire (input, output, internal)
  - SAT-formula: $x_o \wedge G_1 \wedge \dots \wedge G_n$

SAT

- Is a Boolean formula with m operators and n variables satisfiable?

$$\exists x_1 \ldots x_n.\ x_1 \wedge (x_2 \vee \neg x_3) \vee x_4 \ldots$$

- SAT is in NP: solution can be checked by evaluating formula.
  - obviously in polynomial time!
- SAT is NP-hard: show Circuit-SAT $\leq_p$ SAT
  - Introduce one variable per wire (input, output, internal)
  - SAT-formula: $x_o \wedge G_1 \wedge \ldots \wedge G_n$
    - where x_o is variable for output, and for each gate, G_i is

SAT

- Is a Boolean formula with m operators and n variables satisfiable?

$$\exists x_1 \ldots x_n.\ x_1 \wedge (x_2 \vee \neg x_3) \vee x_4 \ldots$$

- SAT is in NP: solution can be checked by evaluating formula.
  - obviously in polynomial time!
- SAT is NP-hard: show Circuit-SAT $\leq_p$ SAT
  - Introduce one variable per wire (input, output, internal)
  - SAT-formula: $x_o \wedge G_1 \wedge \ldots \wedge G_n$
    - where x_o is variable for output, and for each gate, G_i is
    - AND-gate (in x_1, x_2, out x_3): $x_3 \leftrightarrow (x_1 \wedge x_2)$

SAT

- Is a Boolean formula with m operators and n variables satisfiable?

$$\exists x_1 \dots x_n.\ x_1 \wedge (x_2 \vee \neg x_3) \vee x_4 \dots$$

- SAT is in NP: solution can be checked by evaluating formula.
  - obviously in polynomial time!
- SAT is NP-hard: show Circuit-SAT $\leq_p$ SAT
  - Introduce one variable per wire (input, output, internal)
  - SAT-formula: $x_o \wedge G_1 \wedge \dots \wedge G_n$
    - where x_o is variable for output, and for each gate, G_i is
    - AND-gate (in x_1, x_2, out x_3): $x_3 \leftrightarrow (x_1 \wedge x_2)$
    - OR-gate (in x_1, x_2, out x_3): $x_3 \leftrightarrow (x_1 \vee x_2)$

SAT

- Is a Boolean formula with m operators and n variables satisfiable?

$$\exists x_1 \ldots x_n.\ x_1 \wedge (x_2 \vee \neg x_3) \vee x_4 \ldots$$

- SAT is in NP: solution can be checked by evaluating formula.
  - obviously in polynomial time!
- SAT is NP-hard: show Circuit-SAT $\leq_p$ SAT
  - Introduce one variable per wire (input, output, internal)
  - SAT-formula: $x_o \wedge G_1 \wedge \ldots \wedge G_n$
    - where x_o is variable for output, and for each gate, G_i is
    - AND-gate (in x_1, x_2, out x_3): $x_3 \leftrightarrow (x_1 \wedge x_2)$
    - OR-gate (in x_1, x_2, out x_3): $x_3 \leftrightarrow (x_1 \vee x_2)$
    - Inverted input/output: Add a $\neg$!

SAT

- Is a Boolean formula with m operators and n variables satisfiable?

$$\exists x_1 \dots x_n.\ x_1 \wedge (x_2 \vee \neg x_3) \vee x_4 \dots$$

- SAT is in NP: solution can be checked by evaluating formula.
  - obviously in polynomial time!
- SAT is NP-hard: show Circuit-SAT $\leq_p$ SAT
  - Introduce one variable per wire (input, output, internal)
  - SAT-formula: $x_o \wedge G_1 \wedge \dots \wedge G_n$
    - where x_o is variable for output, and for each gate, G_i is
    - AND-gate (in x_1, x_2, out x_3): $x_3 \leftrightarrow (x_1 \wedge x_2)$
    - OR-gate (in x_1, x_2, out x_3): $x_3 \leftrightarrow (x_1 \vee x_2)$
    - Inverted input/output: Add a $\neg$!
    - where $x_1 \leftrightarrow x_2$ is short for $(x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$

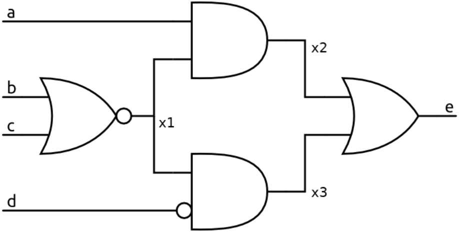
SAT

- Is a Boolean formula with m operators and n variables satisfiable?

$$\exists x_1 \ldots x_n.\ x_1 \wedge (x_2 \vee \neg x_3) \vee x_4 \ldots$$

- SAT is in NP: solution can be checked by evaluating formula.
  - obviously in polynomial time!
- SAT is NP-hard: show Circuit-SAT $\leq_p$ SAT
  - Introduce one variable per wire (input, output, internal)
  - SAT-formula: $x_o \wedge G_1 \wedge \ldots \wedge G_n$
    - where x_o is variable for output, and for each gate, G_i is
    - AND-gate (in x_1, x_2, out x_3): $x_3 \leftrightarrow (x_1 \wedge x_2)$
    - OR-gate (in x_1, x_2, out x_3): $x_3 \leftrightarrow (x_1 \vee x_2)$
    - Inverted input/output: Add a $\neg$!
    - where $x_1 \leftrightarrow x_2$ is short for $(x_1 \wedge x_2) \vee (\neg x_1 \wedge \neg x_2)$
  - can obviously be done in polynomial time
    - constant work per gate and variable!

Circuit-SAT $\leq_p$ SAT



$$
\begin{array}{rr}
 & e \\
\wedge & \neg x_1 \leftrightarrow b \vee c \\
\wedge & x_2 \leftrightarrow a \wedge x_1 \\
\wedge & x_3 \leftrightarrow \neg d \wedge x_1 \\
\wedge & e \leftrightarrow x_2 \vee x_3
\end{array}
$$

3SAT

- Boolean formula in CNF, exactly 3 literals over different variables per clause
  - E.g. $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$

3SAT

- Boolean formula in CNF, exactly 3 literals over different variables per clause
  - E.g. $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$
- Obviously 3SAT is in NP.

3SAT

- Boolean formula in CNF, exactly 3 literals over different variables per clause
  - E.g. $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$
- Obviously 3SAT is in NP.
- To prove it is NP-Hard: Show SAT $\leq_p$ 3SAT

3SAT

- Boolean formula in CNF, exactly 3 literals over different variables per clause
  - E.g. $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$
- Obviously 3SAT is in NP.
- To prove it is NP-Hard: Show SAT $\leq_p$ 3SAT
- Converting SAT formula to 3SAT

3SAT

- Boolean formula in CNF, exactly 3 literals over different variables per clause
  - E.g. $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$
- Obviously 3SAT is in NP.
- To prove it is NP-Hard: Show SAT $\leq_p$ 3SAT
- Converting SAT formula to 3SAT
  1. encode parse tree (same idea as for Circuit-SAT $\leq_p$ SAT)
     - one new variable y_i per node t_i.
     - for root node t_r, add clause y_r
     - for literals: use $y_i = x_i$
     - for node $t_i = AND(t_j, t_k)$, add clause $y_i \leftrightarrow y_j \wedge y_k$
     - similar for OR, NOT

3SAT

- Boolean formula in CNF, exactly 3 literals over different variables per clause
  - E.g. $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$
- Obviously 3SAT is in NP.
- To prove it is NP-Hard: Show SAT $\leq_p$ 3SAT
- Converting SAT formula to 3SAT
  1. encode parse tree (same idea as for Circuit-SAT $\leq_p$ SAT)
     - one new variable y_i per node t_i.
     - for root node t_r, add clause y_r
     - for literals: use $y_i = x_i$
     - for node $t_i = AND(t_j, t_k)$, add clause $y_i \leftrightarrow y_j \wedge y_k$
     - similar for OR, NOT
  2. convert to CNF
     - Convert clauses $x_i \leftrightarrow \ldots$ to CNF
     - e.g. $x_i \leftrightarrow x_j \wedge x_k$ to $(\neg x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$

3SAT

- Boolean formula in CNF, exactly 3 literals over different variables per clause
  - E.g. $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$
- Obviously 3SAT is in NP.
- To prove it is NP-Hard: Show SAT $\leq_p$ 3SAT
- Converting SAT formula to 3SAT
  1. encode parse tree (same idea as for Circuit-SAT $\leq_p$ SAT)
     - one new variable y_i per node t_i.
     - for root node t_r, add clause y_r
     - for literals: use $y_i = x_i$
     - for node $t_i = AND(t_j, t_k)$, add clause $y_i \leftrightarrow y_j \wedge y_k$
     - similar for OR, NOT
  2. convert to CNF
     - Convert clauses $x_i \leftrightarrow \ldots$ to CNF
     - e.g. $x_i \leftrightarrow x_j \wedge x_k$ to $(\neg x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$
  3. eliminate duplicates:
     - remove clauses that contain both, x and $\neg x$
     - remove duplicate literals from clause

3SAT

- Boolean formula in CNF, exactly 3 literals over different variables per clause
  - E.g. $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$
- Obviously 3SAT is in NP.
- To prove it is NP-Hard: Show SAT $\leq_p$ 3SAT
- Converting SAT formula to 3SAT
  1. encode parse tree (same idea as for Circuit-SAT $\leq_p$ SAT)
     - one new variable y_i per node t_i.
     - for root node t_r, add clause y_r
     - for literals: use $y_i = x_i$
     - for node $t_i = AND(t_j, t_k)$, add clause $y_i \leftrightarrow y_j \wedge y_k$
     - similar for OR, NOT
  2. convert to CNF
     - Convert clauses $x_i \leftrightarrow \ldots$ to CNF
     - e.g. $x_i \leftrightarrow x_j \wedge x_k$ to $(\neg x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$
  3. eliminate duplicates:
     - remove clauses that contain both, x and $\neg x$
     - remove duplicate literals from clause
  4. fill clauses to 3 variables
     - a $l_1 \vee l_2$ to $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$ for fresh variable p
     - b l_1 to $(l_1 \vee p) \wedge (l_1 \vee \neg p)$ for fresh variable p, then a)

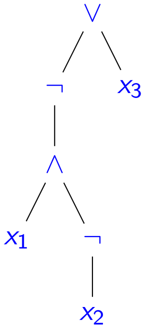
SAT $\leq_p$ 3SAT — Example

SAT Formula: $\neg(x_1 \land \neg x_2) \lor x_3$

SAT $\leq_p$ 3SAT — Example

SAT Formula: $\neg(x_1 \wedge \neg x_2) \vee x_3$

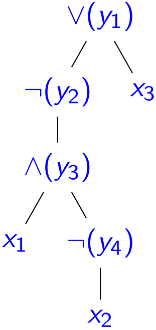
Parse tree:



SAT $\leq_p$ 3SAT — Example

SAT Formula: $\neg(x_1 \wedge \neg x_2) \vee x_3$

Parse tree:

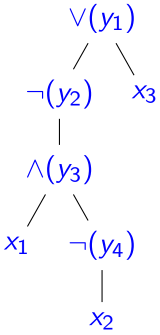


New variables $y_1, \ldots, y_4$.

SAT $\leq_p$ 3SAT — Example

SAT Formula: $\neg(x_1 \wedge \neg x_2) \vee x_3$

Parse tree:



New variables $y_1, \ldots, y_4$.

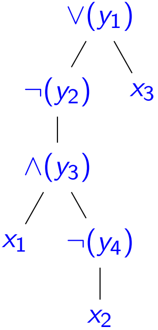
Formula for parse-tree:

$y_1 \wedge (y_1 \leftrightarrow y_2 \vee x_3) \wedge (y_2 \leftrightarrow \neg y_3)$
$\wedge\, (y_3 \leftrightarrow x_1 \wedge y_4) \wedge y_4 \leftrightarrow \neg x_2$

SAT $\leq_p$ 3SAT — Example

SAT Formula: $\neg(x_1 \land \neg x_2) \lor x_3$

Parse tree:



New variables $y_1, \ldots, y_4$.

Formula for parse-tree:

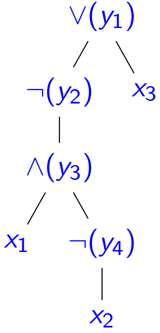
$y_1 \land (y_1 \leftrightarrow y_2 \lor x_3) \land (y_2 \leftrightarrow \neg y_3)$
$\land (y_3 \leftrightarrow x_1 \land y_4) \land y_4 \leftrightarrow \neg x_2$

convert to CNF: $y_1 \land (\neg y_1 \lor y_2 \lor x_3) \land$
$(y_1 \lor \neg y_2) \land (y_1 \lor \neg x_3) \land \ldots$

SAT $\leq_p$ 3SAT — Example

SAT Formula: $\neg(x_1 \wedge \neg x_2) \vee x_3$

Parse tree:



New variables $y_1, \ldots, y_4$.

Formula for parse-tree:
$y_1 \wedge (y_1 \leftrightarrow y_2 \vee x_3) \wedge (y_2 \leftrightarrow \neg y_3)$
$\wedge (y_3 \leftrightarrow x_1 \wedge y_4) \wedge y_4 \leftrightarrow \neg x_2$

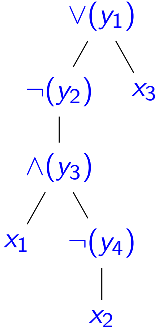
convert to CNF: $y_1 \wedge (\neg y_1 \vee y_2 \vee x_3) \wedge$
$(y_1 \vee \neg y_2) \wedge (y_1 \vee \neg x_3) \wedge \ldots$

(no duplicates)

SAT $\leq_p$ 3SAT — Example

SAT Formula: $\neg(x_1 \wedge \neg x_2) \vee x_3$

Parse tree:



New variables $y_1, \ldots, y_4$.

Formula for parse-tree:

$y_1 \wedge (y_1 \leftrightarrow y_2 \vee x_3) \wedge (y_2 \leftrightarrow \neg y_3)$
$\wedge\ (y_3 \leftrightarrow x_1 \wedge y_4) \wedge y_4 \leftrightarrow \neg x_2$

convert to CNF: $y_1 \wedge (\neg y_1 \vee y_2 \vee x_3) \wedge$
$(y_1 \vee \neg y_2) \wedge (y_1 \vee \neg x_3) \wedge \ldots$

(no duplicates)

fill clauses to 3 variables

$(y_1 \vee p_1 \vee p_2) \wedge (y_1 \vee p_1 \vee \neg p_2)$
$\wedge\ (y_1 \vee \neg p_1 \vee p_2) \wedge (y_1 \vee \neg p_1 \vee \neg p_2)$
$\wedge\ (\neg y_1 \vee y_2 \vee x_3)$
$\wedge\ (y_1 \vee \neg y_2 \vee p_3) \wedge (y_1 \vee \neg y_2 \vee p_3)$
$\wedge \ldots$

Why 3SAT?

- 3-SAT is very restrictive compared to SAT

Why 3SAT?

- 3-SAT is very restrictive compared to SAT
- Bad, if we want to reduce a problem to 3SAT
  - Need to encode problem to very restrictive form

Why 3SAT?

- 3-SAT is very restrictive compared to SAT
- Bad, if we want to reduce a problem to 3SAT
  - Need to encode problem to very restrictive form
- Good, if we want to reduce 3SAT to a problem
  - Only need to encode very special clauses to problem

CLIQUE

- Given an **undirected** graph (V, E).
  - We assume that E is symmetric, i.e., $(u, v) \in E \implies (v, u) \in E$

CLIQUE

- Given an **undirected** graph (V, E).
  - We assume that E is symmetric, i.e., $(u, v) \in E \implies (v, u) \in E$
- A *clique* C is a set of pairwise connected nodes:
  - $C \subseteq V$ and $\forall u, v \in V. u \neq v \implies (u, v) \in E$
  - n-clique: clique of size n

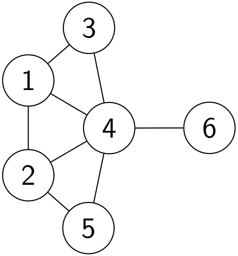
CLIQUE

- Given an **undirected** graph (V, E).
  - We assume that E is symmetric, i.e., $(u, v) \in E \implies (v, u) \in E$
- A *clique* C is a set of pairwise connected nodes:
  - $C \subseteq V$ and $\forall u, v \in V. u \neq v \implies (u, v) \in E$
  - n-clique: clique of size n
- Optimization problem: find a largest clique

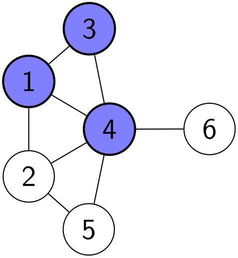
CLIQUE

- Given an **undirected** graph (V, E).
  - We assume that E is symmetric, i.e., $(u, v) \in E \implies (v, u) \in E$
- A *clique* C is a set of pairwise connected nodes:
  - $C \subseteq V$ and $\forall u, v \in V . u \neq v \implies (u, v) \in E$
  - n-clique: clique of size n
- Optimization problem: find a largest clique
- Decision problem: is there a clique of size $\geq k$?

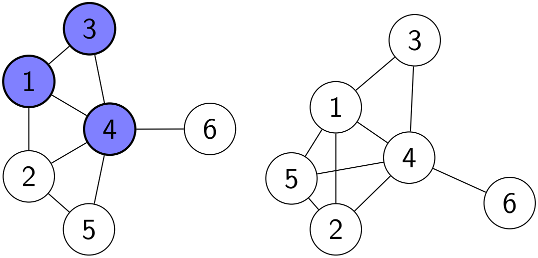
CLIQUE examples



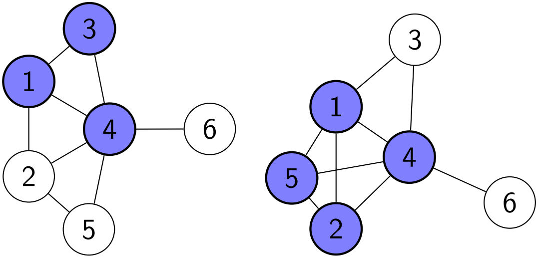
CLIQUE examples



CLIQUE examples



CLIQUE examples



CLIQUE is NP-complete

- Obviously in NP: Given set of nodes can easily be verified to be a clique

CLIQUE is NP-complete

- Obviously in NP: Given set of nodes can easily be verified to be a clique
- NP-hard: Reduction from 3SAT

CLIQUE is NP-complete

- Obviously in NP: Given set of nodes can easily be verified to be a clique
- NP-hard: Reduction from 3SAT
  - given 3SAT problem $(l_1^1 \vee l_1^2 \vee l_1^3) \wedge \ldots \wedge (l_n^1 \vee l_n^2 \vee l_n^3)$

CLIQUE is NP-complete

- Obviously in NP: Given set of nodes can easily be verified to be a clique
- NP-hard: Reduction from 3SAT
  - given 3SAT problem $(l_1^1 \vee l_1^2 \vee l_1^3) \wedge \ldots \wedge (l_n^1 \vee l_n^2 \vee l_n^3)$
  - construct graph:
    - one node per l_i^j: $V = \{u_i^j \mid i \leq n \wedge j \leq 3\}$
    - edges between non-contradicting literals of different clauses: $E = \{(u_i^j, u_{i'}^{j'}) \mid i \neq i' \wedge j \leq 3 \wedge (u_i^j, u_{i'}^{j'}) \text{ not contradicting}\}$
    - contradicting: literals of the form $x, \neg x$.

CLIQUE is NP-complete

- Obviously in NP: Given set of nodes can easily be verified to be a clique
- NP-hard: Reduction from 3SAT
  - given 3SAT problem $(l_1^1 \vee l_1^2 \vee l_1^3) \wedge \ldots \wedge (l_n^1 \vee l_n^2 \vee l_n^3)$
  - construct graph:
    - one node per l_i^j: $V = \{u_i^j \mid i \leq n \wedge j \leq 3\}$
    - edges between non-contradicting literals of different clauses: $E = \{(u_i^j, u_{i'}^{j'}) \mid i \neq i' \wedge j \leq 3 \wedge (u_i^j, u_{i'}^{j'}) \text{ not contradicting}\}$
    - contradicting: literals of the form $x, \neg x$.
  - claim: graph has n-clique, iff formula satisfiable!

CLIQUE is NP-complete

Formula: $(l_1^1 \vee l_1^2 \vee l_1^3) \wedge \ldots \wedge (l_n^1 \vee l_n^2 \vee l_n^3)$
Edge $(u_i^j, u_{i'}^{j'})$ iff $i \neq i'$ and $l_i^j, l_{i'}^{j'}$ non-contr.

CLIQUE is NP-complete

Formula: $(l_1^1 \vee l_1^2 \vee l_1^3) \wedge \ldots \wedge (l_n^1 \vee l_n^2 \vee l_n^3)$
Edge $(u_i^j, u_{i'}^{j'})$ iff $i \neq i'$ and $l_i^j, l_{i'}^{j'}$ non-contr.
Assume solution to formula.

- each clause has at least one satisfied literal. n satisfied literals.
- each satisfied literal connected with all other satisfied literals of different clauses
- clique of size n

CLIQUE is NP-complete

Formula: $(l_1^1 \vee l_1^2 \vee l_1^3) \wedge \ldots \wedge (l_n^1 \vee l_n^2 \vee l_n^3)$
Edge $(u_i^j, u_{i'}^{j'})$ iff $i \neq i'$ and $l_i^j, l_{i'}^{j'}$ non-contr.
Assume solution to formula.

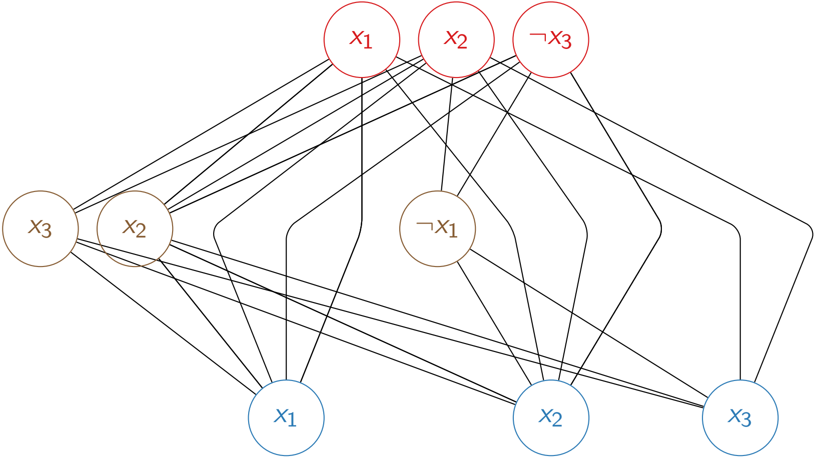
- each clause has at least one satisfied literal. n satisfied literals.
- each satisfied literal connected with all other satisfied literals of different clauses
- clique of size n

Assume clique of size n.

- must involve literals of different clauses
- which are non-contradictory
- setting them to true yields solution

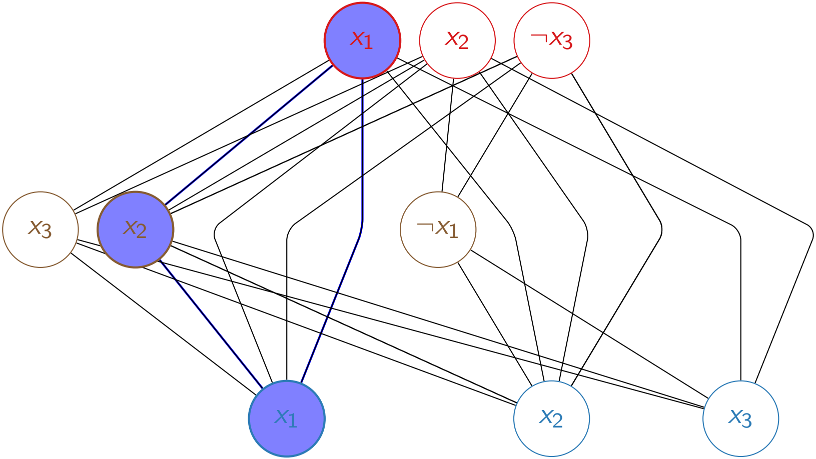
3SAT $\leq_p$ CLIQUE Example

Formula: $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$



3SAT $\leq_p$ CLIQUE Example

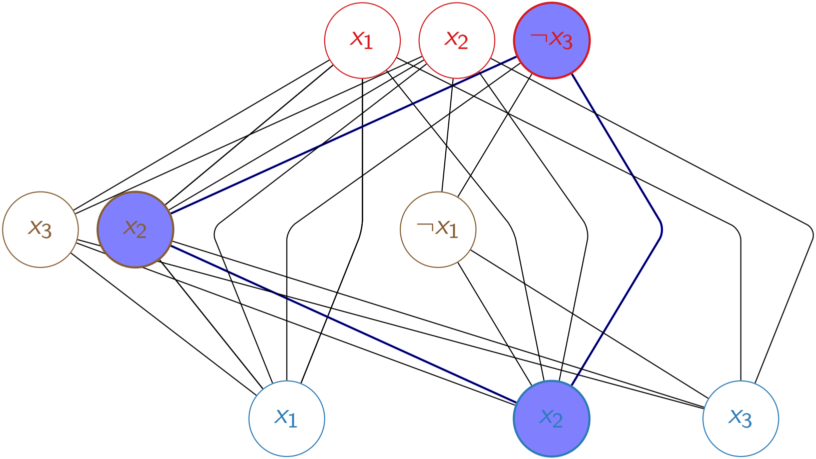
Formula: $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$



Solution: $x_1 = \top, x_2 = \top, x_3 = ?$

3SAT $\leq_p$ CLIQUE Example

Formula: $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_3)$



Solution: $x_1 =?, x_2 = \top, x_3 = \bot$

Conclusions

- NP and NP-complete problems
  - no poly-time algorithms known for NP-hard problems
  - if you encounter one: special case?, approximation?
- Prove that problem is NP-complete:
  - in NP: show poly-time certification
  - NP-hard: reduce other NP-hard problem to it
- Many realistic problems NP-hard
  - You'll come across a few more in this lecture
  - Knapsack, Integer Linear Programming, ...