CSI - 3105 Design & Analysis of Algorithms
Course 19

Jean-Lou De Carufel

Fall 2019

Theorem

The relation $\leq_P$ is transitive:

$$L \leq_P L' \quad \text{and} \quad L' \leq_P L'' \qquad \Longrightarrow \qquad L \leq_P L''$$

PROOF:

Theorem

The relation $\leq_P$ is transitive:

$$L \leq_P L' \quad \text{and} \quad L' \leq_P L'' \quad \Longrightarrow \quad L \leq_P L''$$

PROOF:

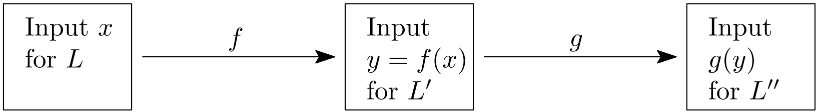
$$\boxed{\begin{array}{l}\text{Input } x\\ \text{for } L\end{array}} \xrightarrow{\quad f \quad} \boxed{\begin{array}{l}\text{Input}\\ y = f(x)\\ \text{for } L'\end{array}} \xrightarrow{\quad g \quad} \boxed{\begin{array}{l}\text{Input}\\ g(y)\\ \text{for } L''\end{array}}$$

Theorem

The relation $\leq_P$ is transitive:

$$L \leq_P L' \quad \text{and} \quad L' \leq_P L'' \quad \Longrightarrow \quad L \leq_P L''$$

PROOF:



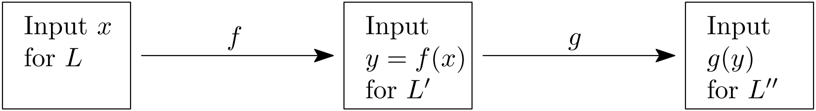
$$x \in L \quad \Longleftrightarrow \quad y = f(x) \in L' \quad \Longleftrightarrow \quad g(y) \in L''$$

Theorem

The relation $\leq_P$ is transitive:

$$L \leq_P L' \quad \text{and} \quad L' \leq_P L'' \quad \Longrightarrow \quad L \leq_P L''$$

PROOF:



$$x \in L \iff y = f(x) \in L' \iff g(y) \in L''$$

Thus,

$$x \in L \iff g(f(x)) \in L''$$

Theorem

The relation $\leq_P$ is transitive:

$$L \leq_P L' \quad \text{and} \quad L' \leq_P L'' \quad \Longrightarrow \quad L \leq_P L''$$

PROOF:

$$\boxed{\begin{array}{l}\text{Input } x\\ \text{for } L\end{array}} \xrightarrow{\quad f \quad} \boxed{\begin{array}{l}\text{Input}\\ y = f(x)\\ \text{for } L'\end{array}} \xrightarrow{\quad g \quad} \boxed{\begin{array}{l}\text{Input}\\ g(y)\\ \text{for } L''\end{array}}$$

$$x \in L \quad \Longleftrightarrow \quad y = f(x) \in L' \quad \Longleftrightarrow \quad g(y) \in L''$$

Thus,

$$x \in L \quad \Longleftrightarrow \quad g(f(x)) \in L''$$

The reduction from L to L'' is given by the function $g \circ f$.

Theorem

The relation $\leq_P$ is transitive:

$$L \leq_P L' \quad \text{and} \quad L' \leq_P L'' \quad \Longrightarrow \quad L \leq_P L''$$

PROOF:

$$\boxed{\begin{array}{l}\text{Input } x\\ \text{for } L\end{array}} \xrightarrow{\quad f \quad} \boxed{\begin{array}{l}\text{Input}\\ y = f(x)\\ \text{for } L'\end{array}} \xrightarrow{\quad g \quad} \boxed{\begin{array}{l}\text{Input}\\ g(y)\\ \text{for } L''\end{array}}$$

$$x \in L \quad \Longleftrightarrow \quad y = f(x) \in L' \quad \Longleftrightarrow \quad g(y) \in L''$$

Thus,

$$x \in L \quad \Longleftrightarrow \quad g(f(x)) \in L''$$

The reduction from L to L'' is given by the function $g \circ f$. Given x, $(g \circ f)(x) = g(f(x))$ can be computed in time that is polynomial in the length of x (do you see why?)

§6.4 *NP*-Hard and *NP*-Complete

§6.4 *NP*-Hard and *NP*-Complete

The language L is *NP-Hard* if

- For all $L' \in NP$, $L' \leq_P L$.

§6.4 NP-Hard and NP-Complete

The language L is *NP-Hard* if

- For all $L' \in NP$, $L' \leq_P L$.

The language L is *NP-Complete* if

- $L \in NP$
- and for all $L' \in NP$, $L' \leq_P L$.

§6.4 NP-Hard and NP-Complete

The language L is *NP-Hard* if

- For all $L' \in NP$, $L' \leq_P L$.

The language L is *NP-Complete* if

- $L \in NP$
- and for all $L' \in NP$, $L' \leq_P L$.

Intuitively, this means that L belongs to the most difficult problems in NP.

This is what we were looking for in §6.2.

Theorem

Assume that L is NP-Complete. Then

$$L \in P \iff P = NP.$$

Theorem

Assume that L is NP-Complete. Then

$$L \in P \iff P = NP.$$

Intuition:

- If $L \in P$, L is easy.
- L is NP-Complete, so L belongs to the most difficult problems in NP.
- Well... if the most difficult problem in NP turns out to be easy, then all problems in NP are easy!

Theorem

Assume that L is NP-Complete. Then

$$L \in P \iff P = NP.$$

Intuition:

- If $L \in P$, L is easy.
- L is NP-Complete, so L belongs to the most difficult problems in NP.
- Well... if the most difficult problem in NP turns out to be easy, then all problems in NP are easy!

PROOF: $[\Longleftarrow]$ Assume that $P = NP$.

Theorem

Assume that L is NP-Complete. Then

$$L \in P \iff P = NP.$$

Intuition:

- If $L \in P$, L is easy.
- L is NP-Complete, so L belongs to the most difficult problems in NP.
- Well... if the most difficult problem in NP turns out to be easy, then all problems in NP are easy!

PROOF: $[\Longleftarrow]$ Assume that $P = NP$.

Since L is NP-Complete, $L \in NP$. Thus, $L \in P$.

Theorem

Assume that L is NP-Complete. Then

$$L \in P \iff P = NP.$$

Intuition:

- If $L \in P$, L is easy.
- L is NP-Complete, so L belongs to the most difficult problems in NP.
- Well... if the most difficult problem in NP turns out to be easy, then all problems in NP are easy!

PROOF: $[\Longleftarrow]$ Assume that $P = NP$.

Since L is NP-Complete, $L \in NP$. Thus, $L \in P$.

$[\Longrightarrow]$ Assume that $L \in P$. We have to show that $P = NP$.

Theorem

Assume that L is NP-Complete. Then

$$L \in P \iff P = NP.$$

Intuition:

- If $L \in P$, L is easy.
- L is NP-Complete, so L belongs to the most difficult problems in NP.
- Well... if the most difficult problem in NP turns out to be easy, then all problems in NP are easy!

PROOF: $[\Longleftarrow]$ Assume that $P = NP$.

Since L is NP-Complete, $L \in NP$. Thus, $L \in P$.

$[\Longrightarrow]$ Assume that $L \in P$. We have to show that $P = NP$.

We know that $P \subseteq NP$ (one of the previous theorems).

Theorem

Assume that L is NP-Complete. Then

$$L \in P \iff P = NP.$$

Intuition:

- If $L \in P$, L is easy.
- L is NP-Complete, so L belongs to the most difficult problems in NP.
- Well... if the most difficult problem in NP turns out to be easy, then all problems in NP are easy!

PROOF: $[\Longleftarrow]$ Assume that $P = NP$.

Since L is NP-Complete, $L \in NP$. Thus, $L \in P$.

$[\Longrightarrow]$ Assume that $L \in P$. We have to show that $P = NP$.

We know that $P \subseteq NP$ (one of the previous theorems). To show that $NP \subset P$, let $L' \in NP$.

Theorem

Assume that L is NP-Complete. Then

$$L \in P \iff P = NP.$$

Intuition:

- If $L \in P$, L is easy.
- L is NP-Complete, so L belongs to the most difficult problems in NP.
- Well... if the most difficult problem in NP turns out to be easy, then all problems in NP are easy!

PROOF: $[\Longleftarrow]$ Assume that $P = NP$.

Since L is NP-Complete, $L \in NP$. Thus, $L \in P$.

$[\Longrightarrow]$ Assume that $L \in P$. We have to show that $P = NP$.

We know that $P \subseteq NP$ (one of the previous theorems). To show that $NP \subset P$, let $L' \in NP$.
Since L is NP-Complete, $L' \leq_P L$.

Theorem

Assume that L is NP-Complete. Then

$$L \in P \iff P = NP.$$

Intuition:

- If $L \in P$, L is easy.
- L is NP-Complete, so L belongs to the most difficult problems in NP.
- Well... if the most difficult problem in NP turns out to be easy, then all problems in NP are easy!

PROOF: [$\Longleftarrow$] Assume that $P = NP$.

Since L is NP-Complete, $L \in NP$. Thus, $L \in P$.

[$\Longrightarrow$] Assume that $L \in P$. We have to show that $P = NP$.

We know that $P \subseteq NP$ (one of the previous theorems). To show that $NP \subset P$, let $L' \in NP$.
Since L is NP-Complete, $L' \leq_P L$. Since $L \in P$, then $L' \in P$ (one of the previous theorems).

Theorem

$$\left.\begin{array}{l} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Theorem

$$\left.\begin{array}{r} L \textit{ is NP-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \textit{ is NP-Complete}$$

Intuition:

- L is NP-Complete, so L belongs to the most difficult problems in NP.
- L' is at least as difficult as L.
- Then L' also belongs to the most difficult problems in NP.

Theorem

$$\left.\begin{array}{l} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Intuition:

- L is NP-Complete, so L belongs to the most difficult problems in NP.
- L' is at least as difficult as L.
- Then L' also belongs to the most difficult problems in NP.

PROOF: To show that L' is NP-Complete, we have to show

Theorem

$$\left.\begin{array}{l} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Intuition:

- L is NP-Complete, so L belongs to the most difficult problems in NP.
- L' is at least as difficult as L.
- Then L' also belongs to the most difficult problems in NP.

PROOF: To show that L' is NP-Complete, we have to show

- $L' \in NP$.
- For each $L'' \in NP$, we must have $L'' \leq_P L'$.

Theorem

$$\left.\begin{array}{r} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Intuition:

- L is NP-Complete, so L belongs to the most difficult problems in NP.
- L' is at least as difficult as L.
- Then L' also belongs to the most difficult problems in NP.

PROOF: To show that L' is NP-Complete, we have to show

- $L' \in NP$. This is given in the statement of the theorem!
- For each $L'' \in NP$, we must have $L'' \leq_P L'$.

Theorem

$$\left.\begin{array}{l} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Intuition:

- L is NP-Complete, so L belongs to the most difficult problems in NP.
- L' is at least as difficult as L.
- Then L' also belongs to the most difficult problems in NP.

PROOF: To show that L' is NP-Complete, we have to show

- $L' \in NP$. This is given in the statement of the theorem!
- For each $L'' \in NP$, we must have $L'' \leq_P L'$. Why is this true?

Theorem

$$\left.\begin{array}{r} L \textit{ is NP-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \textit{ is NP-Complete}$$

Intuition:

- L is NP-Complete, so L belongs to the most difficult problems in NP.
- L' is at least as difficult as L.
- Then L' also belongs to the most difficult problems in NP.

PROOF: To show that L' is NP-Complete, we have to show

- $L' \in NP$. This is given in the statement of the theorem!
- For each $L'' \in NP$, we must have $L'' \leq_P L'$. Why is this true?

  Since L is NP-Complete, $L'' \leq_P L$.

Theorem

$$\left.\begin{array}{r} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Intuition:

- L is NP-Complete, so L belongs to the most difficult problems in NP.
- L' is at least as difficult as L.
- Then L' also belongs to the most difficult problems in NP.

PROOF: To show that L' is NP-Complete, we have to show

- $L' \in NP$. This is given in the statement of the theorem!
- For each $L'' \in NP$, we must have $L'' \leq_P L'$. Why is this true?

  Since L is NP-Complete, $L'' \leq_P L$. We are given $L \leq_P L'$.

Theorem

$$\left.\begin{array}{r} L \textit{ is } NP\textit{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \textit{ is } NP\textit{-Complete}$$

Intuition:

- L is NP-Complete, so L belongs to the most difficult problems in NP.
- L' is at least as difficult as L.
- Then L' also belongs to the most difficult problems in NP.

PROOF: To show that L' is NP-Complete, we have to show

- $L' \in NP$. This is given in the statement of the theorem!
- For each $L'' \in NP$, we must have $L'' \leq_P L'$. Why is this true?

  Since L is NP-Complete, $L'' \leq_P L$. We are given $L \leq_P L'$. Then, by transitivity, we have $L'' \leq_P L'$. $\square$

So What?!

Theorem

$$\left.\begin{array}{r} L \text{ is } NP\text{-}Complete \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-}Complete$$

Here is how to use this theorem:

So What?!

Theorem

$$\left.\begin{array}{l} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Here is how to use this theorem: To show that L' is NP-Complete,

So What?!

Theorem

$$\left.\begin{array}{l} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Here is how to use this theorem: To show that L' is NP-Complete,

1. Show that $L' \in NP$.

So What?!

Theorem

$$\left.\begin{array}{l} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Here is how to use this theorem: To show that L' is NP-Complete,

1. Show that $L' \in NP$.
2. Look for a problem L that is "similar" to L' and that is known to be NP-Complete.

So What?!

Theorem

$$\left.\begin{array}{l} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Here is how to use this theorem: To show that L' is NP-Complete,

1. Show that $L' \in NP$.
2. Look for a problem L that is "similar" to L' and that is known to be NP-Complete.
3. Show that $L \leq_P L'$.

So What?!

Theorem

$$\left.\begin{array}{l} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Here is how to use this theorem: To show that L' is NP-Complete,

1. Show that $L' \in NP$.
2. Look for a problem L that is "similar" to L' and that is known to be NP-Complete.
3. Show that $L \leq_P L'$.

In order to apply this, we need *a first* NP-Complete problem.

We need *a first* NP-Complete problem. So we need one language L in NP such that...

We need *a first* NP-Complete problem. So we need one language L in NP such that...

$$HAMCYCLE \leq_P L$$

We need *a first* NP-Complete problem. So we need one language L in NP such that...

$$HAMCYCLE \leq_P L$$
$$TSP \leq_P L$$

We need *a first* NP-Complete problem. So we need one language L in NP such that...

$$HAMCYCLE \leq_P L$$
$$TSP \leq_P L$$
$$SUBSET - SUM \leq_P L$$

We need *a first* NP-Complete problem. So we need one language L in NP such that...

$$
\begin{aligned}
HAMCYCLE &\leq_P L \\
TSP &\leq_P L \\
SUBSET-SUM &\leq_P L \\
CLIQUE &\leq_P L
\end{aligned}
$$

We need *a first* NP-Complete problem. So we need one language L in NP such that...

$$\begin{aligned} HAMCYCLE &\leq_P L \\ TSP &\leq_P L \\ SUBSET-SUM &\leq_P L \\ CLIQUE &\leq_P L \\ INDEP-SET &\leq_P L \end{aligned}$$

We need *a first* NP-Complete problem. So we need one language L in NP such that...

$$\begin{aligned} HAMCYCLE &\leq_P L \\ TSP &\leq_P L \\ SUBSET-SUM &\leq_P L \\ CLIQUE &\leq_P L \\ INDEP-SET &\leq_P L \\ VERTEX-COVER &\leq_P L \end{aligned}$$

We need *a first* NP-Complete problem. So we need one language L in NP such that...

$$
\begin{aligned}
HAMCYCLE &\leq_P L \\
TSP &\leq_P L \\
SUBSET-SUM &\leq_P L \\
CLIQUE &\leq_P L \\
INDEP-SET &\leq_P L \\
VERTEX-COVER &\leq_P L \\
3SAT &\leq_P L
\end{aligned}
$$

We need *a first* NP-Complete problem. So we need one language L in NP such that...

$$\begin{aligned} HAMCYCLE &\leq_P L \\ TSP &\leq_P L \\ SUBSET-SUM &\leq_P L \\ CLIQUE &\leq_P L \\ INDEP-SET &\leq_P L \\ VERTEX-COVER &\leq_P L \\ 3SAT &\leq_P L \\ &\vdots \end{aligned}$$

We need *a first* NP-Complete problem. So we need one language L in NP such that...

$$\begin{aligned} HAMCYCLE &\leq_P L \\ TSP &\leq_P L \\ SUBSET-SUM &\leq_P L \\ CLIQUE &\leq_P L \\ INDEP-SET &\leq_P L \\ VERTEX-COVER &\leq_P L \\ 3SAT &\leq_P L \\ &\vdots \end{aligned}$$

It is not even clear whether such a problem exists!!!

1971 : Stephen Cook proved that SAT is NP-Complete.

1972 : (independently in Russia) Leonid Levin proved that a certain tiling problem is NP-Complete.

1971 : Stephen Cook proved that SAT is NP-Complete.

1972 : (independently in Russia) Leonid Levin proved that a certain tiling problem is NP-Complete.

We will show that CIRCUIT-SAT is NP-Complete.

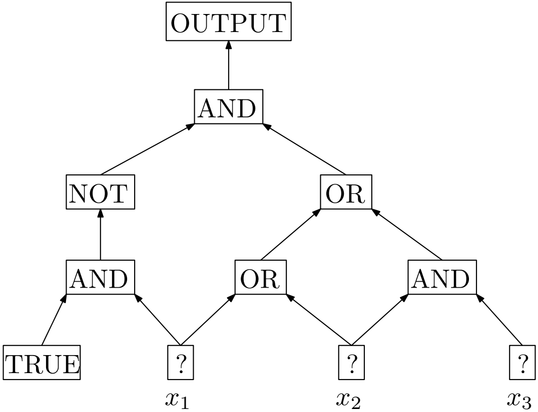
CIRCUIT-SAT

input: A Boolean circuit.

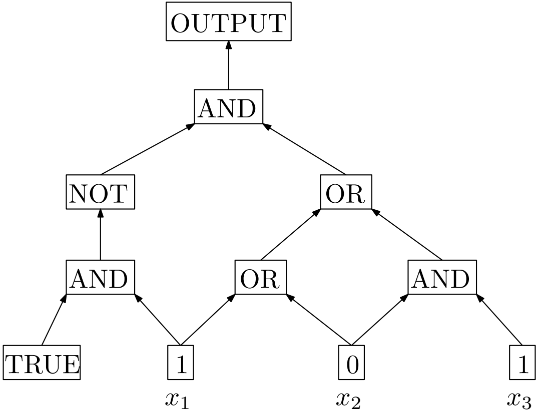
- Directed acyclic graph, where vertices are gates
- AND-gates and OR-gates have indegree 2
- Known input gates have indegree 0 and are labeled TRUE or FALSE.
- Unknown input gates have indegree 0 and are labeled "?".
- There is one output gate (whose outdegree is 0).

question: Is it possible to assign a truth-value to each unknown input gate, such that the output of the circuit is TRUE?

CIRCUIT-SAT

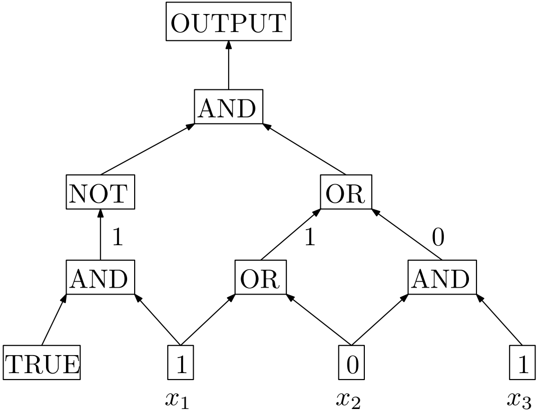


CIRCUIT-SAT



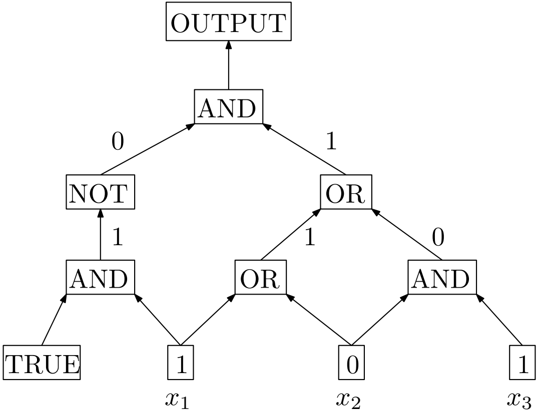
With $x_1 = 1$, $x_2 = 0$, $x_3 = 1$,

CIRCUIT-SAT



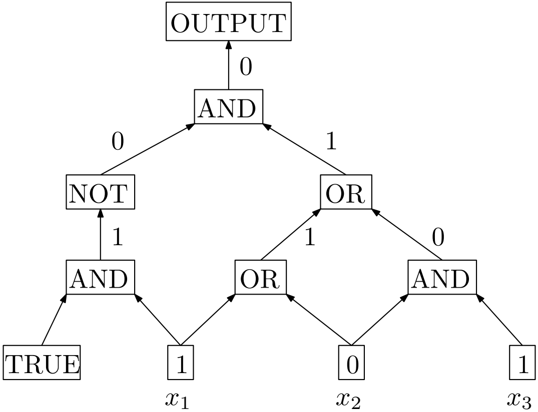
With $x_1 = 1$, $x_2 = 0$, $x_3 = 1$,

CIRCUIT-SAT



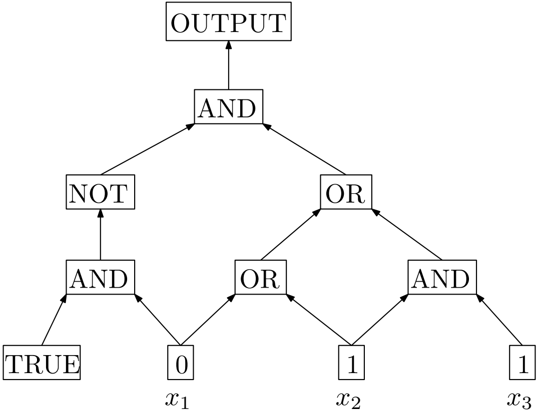
With $x_1 = 1$, $x_2 = 0$, $x_3 = 1$,

CIRCUIT-SAT



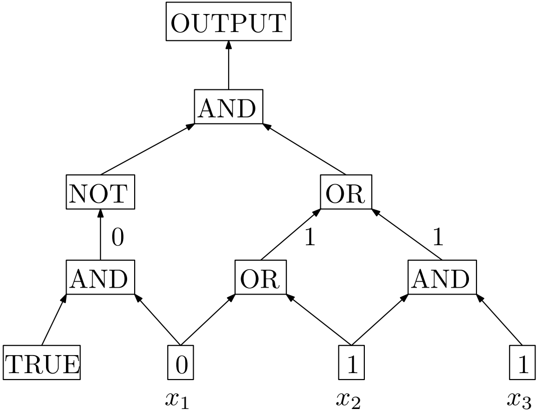
With $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, we get $OUTPUT = 0$.

CIRCUIT-SAT



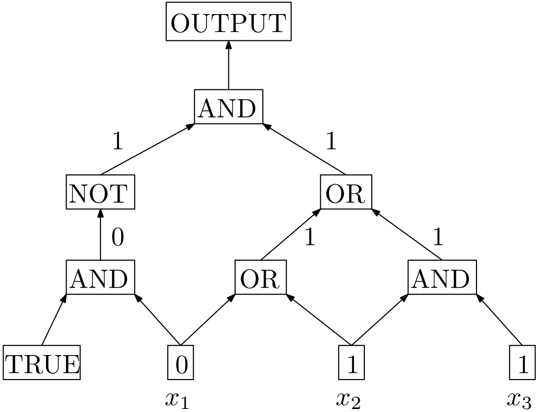
With $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, we get $OUTPUT = 0$.
With $x_1 = 0$, $x_2 = 1$, $x_3 = 1$,

CIRCUIT-SAT



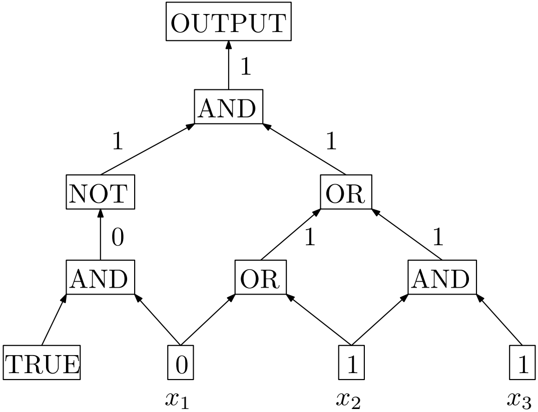
With $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, we get $OUTPUT = 0$.
With $x_1 = 0$, $x_2 = 1$, $x_3 = 1$,

CIRCUIT-SAT



With $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, we get $OUTPUT = 0$.
With $x_1 = 0$, $x_2 = 1$, $x_3 = 1$,

CIRCUIT-SAT



With $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, we get $OUTPUT = 0$.
With $x_1 = 0$, $x_2 = 1$, $x_3 = 1$, we get $OUTPUT = 1$.

$$\begin{aligned} CIRCUIT-SAT = \{B \mid\ & B \text{ is a Boolean circuit for which} \\ & \text{there exist truth values for the unknown} \\ & \text{input gates such that the ouput of } B \text{ is TRUE}\} \end{aligned}$$

$$\begin{aligned} CIRCUIT - SAT = \{B \mid\ & B \text{ is a Boolean circuit for which} \\ & \text{there exist truth values for the unknown} \\ & \text{input gates such that the ouput of } B \text{ is TRUE}\} \end{aligned}$$

To show that CIRCUIT-SAT is NP-Complete, we have to show two things:

$$\begin{aligned} CIRCUIT - SAT = \{B \mid\ & B \text{ is a Boolean circuit for which} \\ & \text{there exist truth values for the unknown} \\ & \text{input gates such that the ouput of } B \text{ is TRUE}\} \end{aligned}$$

To show that CIRCUIT-SAT is NP-Complete, we have to show two things:

1. CIRCUIT-SAT is in NP.

$$CIRCUIT - SAT = \{B \mid B \text{ is a Boolean circuit for which there exist truth values for the unknown input gates such that the ouput of } B \text{ is TRUE}\}$$

To show that CIRCUIT-SAT is NP-Complete, we have to show two things:

1. CIRCUIT-SAT is in NP.
2. Show that for all L in NP, $L \leq_P CIRCUIT - SAT$

$CIRCUIT - SAT = \{B \mid$ B is a Boolean circuit for which
there exist truth values for the unknown
input gates such that the ouput of B is TRUE$\}$

To show that CIRCUIT-SAT is NP-Complete, we have to show two things:

1. CIRCUIT-SAT is in NP.
2. Show that for all L in NP, $L \leq_P CIRCUIT - SAT$

The first item is easy:

CIRCUIT-SAT is in NP:

CIRCUIT-SAT is in *NP*:

Certificate: sequence of truth values for the unknown input gates.

CIRCUIT-SAT is in NP:

Certificate: sequence of truth values for the unknown input gates.

Verification: evaluate the circuit

CIRCUIT-SAT is in NP:

Certificate: sequence of truth values for the unknown input gates.

Verification: evaluate the circuit (evaluate the gates in topological order.)

How do we show that for all L in NP, $L \leq_P CIRCUIT-SAT$?

How do we show that for all L in NP, $L \leq_P CIRCUIT - SAT$?

Let $L \in NP$. We need a function f such that

How do we show that for all L in NP, $L \leq_P CIRCUIT-SAT$?

Let $L \in NP$. We need a function f such that

1. f takes any input x for L and produces an input $B = f(x)$ for $CIRCUIT-SAT$.

How do we show that for all L in NP, $L \leq_P CIRCUIT - SAT$?

Let $L \in NP$. We need a function f such that

1. f takes any input x for L and produces an input $B = f(x)$ for $CIRCUIT - SAT$.
2. $x \in L \iff B \in CIRCUIT - SAT$

How do we show that for all L in NP, $L \leq_P CIRCUIT - SAT$?

Let $L \in NP$. We need a function f such that

1. f takes any input x for L and produces an input $B = f(x)$ for $CIRCUIT - SAT$.
2. $x \in L \iff B \in CIRCUIT - SAT$
3. The time time to compute B is poylnomial in the length of x.

We know that $L \in NP$. So there is a verification algorithm V such that

We know that $L \in NP$. So there is a verification algorithm V such that

- The input to V is (x, y), where x is an input for L and y is a certificate.
- For every input x to L,

$$x \in L \iff \text{there exists a certificate } y \text{ such that}$$

$\cdot\ |y| \leq |x|^c$,

$\cdot\ V(x, y)$ returns YES

$\cdot$ and the running time of $V(x, y)$ is at most $|x|^{c'}$.

We now define the function f.

We now define the function f. Let x be an input for L. Define a new algorithm V_x:

We now define the function f. Let x be an input for L. Define a new algorithm V_x:

- input is a string y of length at most $|x|^c$
- $V_x(y)$ runs $V(x, y)$
- If $V(x, y)$ terminates in at most $|x|^{c'}$ steps, then $V_x(y)$ terminates and returns the output of $V(x, y)$.
- If $V(x, y)$ has not terminated after $|x|^{c'}$ steps, then $V_x(y)$ terminates and returns NO.

We now define the function f. Let x be an input for L. Define a new algorithm V_x:

- input is a string y of length at most $|x|^c$
- $V_x(y)$ runs $V(x, y)$
- If $V(x, y)$ terminates in at most $|x|^{c'}$ steps, then $V_x(y)$ terminates and returns the output of $V(x, y)$.
- If $V(x, y)$ has not terminated after $|x|^{c'}$ steps, then $V_x(y)$ terminates and returns NO.

Observe:

- Running time of Algorithm V_x is at most $|x|^{c'}$.
-

$$x \in L \iff \text{there exists an input } y \text{ for Algorithm } V_x \text{ such that } V_x(y) \text{ returns YES}$$

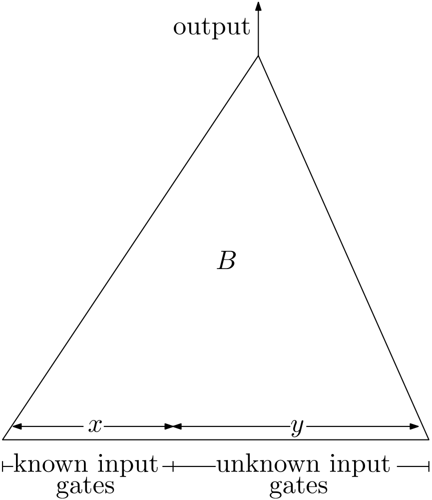
The algorithm V_x is a program that can be run on a computer.

The algorithm V_x is a program that can be run on a computer.

Therefore, V_x can be represented by a Boolean circuit B.

The algorithm V_x is a program that can be run on a computer.

Therefore, V_x can be represented by a Boolean circuit B.



size of B:

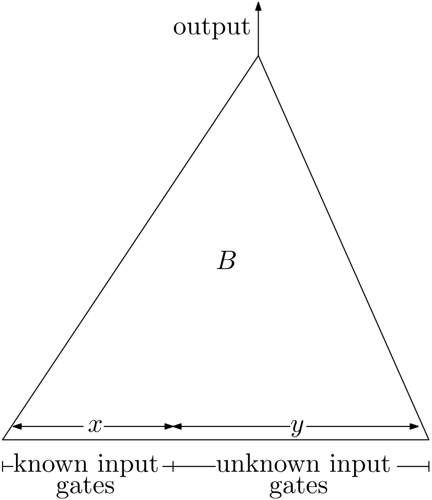
$$O\left(|x| + |y| + |x|^{c'}\right)$$

$$= O\left(|x| + |x|^{c} + |x|^{c'}\right)$$

$$= \text{polynomial in } x!$$

The algorithm V_x is a program that can be run on a computer.

Therefore, V_x can be represented by a Boolean circuit B.



size of B:

$$O\left(|x| + |y| + |x|^{c'}\right)$$

$$= O\left(|x| + |x|^{c} + |x|^{c'}\right)$$

$$= \text{polynomial in } x!$$

The functions f maps x to B!

We have

$$x \in L \iff \text{There exists } y \text{ such that } V_x(y) \text{ returns TRUE}$$

$$\text{(definition of } V_x\text{)}$$

We have

$$
\begin{aligned}
x \in L \quad &\iff \quad \text{There exists } y \text{ such that } V_x(y) \text{ returns TRUE} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\text{definition of } V_x) \\
&\iff \quad \text{There exists } y \text{ such that the output of } B \text{ is TRUE} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\text{definition of } B)
\end{aligned}
$$

We have

$$
\begin{aligned}
x \in L \quad &\iff \quad \text{There exists } y \text{ such that } V_x(y) \text{ returns TRUE} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(definition of } V_x\text{)} \\
&\iff \quad \text{There exists } y \text{ such that the output of } B \text{ is TRUE} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(definition of } B\text{)} \\
&\iff \quad B \in CIRCUIT-SAT
\end{aligned}
$$

We have

$$
\begin{aligned}
x \in L \quad &\iff \quad \text{There exists } y \text{ such that } V_x(y) \text{ returns TRUE} \\
&\hspace{8cm} (\text{definition of } V_x) \\
&\iff \quad \text{There exists } y \text{ such that the output of } B \text{ is TRUE} \\
&\hspace{9cm} (\text{definition of } B) \\
&\iff \quad B \in CIRCUIT - SAT
\end{aligned}
$$

Conclusion: CIRCUIT-SAT is NP-COMPLETE!

Remember the following theorem (refer to Course 20).

Theorem

$$\left.\begin{array}{r} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Remember the following theorem (refer to Course 20).

Theorem

$$\left.\begin{array}{r} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Now we can start using this theorem to prove that other problems are NP-Complete.

Remember the following theorem (refer to Course 20).

Theorem

$$\left.\begin{array}{r} L \textit{ is } NP\textit{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \textit{ is } NP\textit{-Complete}$$

Now we can start using this theorem to prove that other problems are NP-Complete.

At this moment, we know that $CIRCUIT - SAT$ is NP-Complete.

Remember the following theorem (refer to Course 20).

Theorem

$$\left.\begin{array}{r} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Now we can start using this theorem to prove that other problems are NP-Complete.

At this moment, we know that $CIRCUIT - SAT$ is NP-Complete. We will prove that 3SAT is NP-Complete.

Remember the following theorem (refer to Course 20).

Theorem

$$\left.\begin{array}{l} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Now we can start using this theorem to prove that other problems are *NP*-Complete.

At this moment, we know that $CIRCUIT - SAT$ is *NP*-Complete. We will prove that 3SAT is *NP*-Complete. It is sufficient to show that

1. 3SAT is in *NP*.
2. $CIRCUIT - SAT \leq_P 3SAT$

Remember the following theorem (refer to Course 20).

Theorem

$$\left.\begin{array}{r} L \text{ is } NP\text{-Complete} \\ L \leq_P L' \\ L' \in NP \end{array}\right\} \implies L' \text{ is } NP\text{-Complete}$$

Now we can start using this theorem to prove that other problems are NP-Complete.

At this moment, we know that $CIRCUIT - SAT$ is NP-Complete. We will prove that 3SAT is NP-Complete. It is sufficient to show that

1. 3SAT is in NP.
2. $CIRCUIT - SAT \leq_P 3SAT$

The first item is easy: for a given truth-assignment of the variables, we can verify in polynomial time if the Boolean formula is true.

It remains to show that $CIRCUIT - SAT \leq_P 3SAT$.

It remains to show that $CIRCUIT - SAT \leq_P 3SAT$. We need a function f such that

1. f transforms any input (Boolean circuit) B for $CIRCUIT - SAT$ and produces an input $\phi = f(B)$ (Boolean formula) for 3SAT.
2. There exist truth-values for the unknown input gates such that B's output is true

   $$\iff$$

   There exist truth-values for the variables such that ϕ is true
3. $\phi = f(B)$ can be computed in time that is polynomial in the size of B.