NP-Completeness II

Chapter 34

Lecture Overview

- Formal Languages as a setting for P and NP
- NP-Completeness
- Boolean Combinational Circuits
- CIRCUIT-SAT is NPC
- Formula Satisfiability (SAT)
- SAT is NPC

Formal Language Setting

The text introduces a formal language setting for P and NP. Let Σ be a finite alphabet (set of characters); then Σ* denotes the set of strings one can form with characters of the alphabet.

A (formal) language L is a subset of Σ*.

We could have

- Σ = {a, b} and L = {w | w ends with aba}.
- Σ = ASCII and L = {p | p is a correct C program}.
- Σ = {0, 1} and L = {p | p is a correct ASCII-encoded C program}

Formal Language Setting

In fact, by encoding all characters of Σ into binary, we can consider all languages to have the alphabet {0, 1}.

Now we can consider problems like Hamiltonian Cycle as formal languages:

HAM-CYCLE = { w | w is the encoding of a graph that has a Hamiltonian cycle.}

A language L can be recognized in time T(n) if there is a program that, given an input string of length n, can determine if that string is in L in time T(n).

Formal Language Setting

- P is then the set of all languages that can be recognized in polynomial time.
- NP is the set of all languages L that can be verified in polynomial time. A language can be verified in time O(T(n)) if given any string s of length n, there is a certificate t such that a program given s and t can tell, in time O(T(n)), whether s is in L.

Reductions

A language L_1 is called polynomial-time reducible to language L_2, written $L_1 \leq_P L_2$, if there is a polynomial-time computable function f such that:

$x \in L_1$ iff $f(x) \in L_2$.

Lemma. If $L_1, L_2 \subseteq \{0, 1\}^*$ are languages such that $L_1 \leq_P L_2$, then $L_2 \in P$ implies $L_1 \in P$.

NP-Completeness

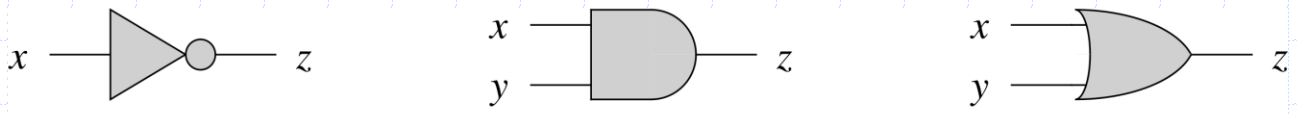
A language $L \subseteq \{0, 1\}^*$ is NP-Complete if

1. $L \in NP$
2. $L' \leq_P L$ for every $L' \in NP$

A language satisfying condition 2 but not condition 1 is called NP-Hard.

Boolean Combinational Circuits

A Boolean combinational circuit is a circuit composed of simple Boolean combinational gates, which for our purposes are NOT, AND, and OR gates.



x	$\neg x$
0	1
1	0

(a)

x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

(b)

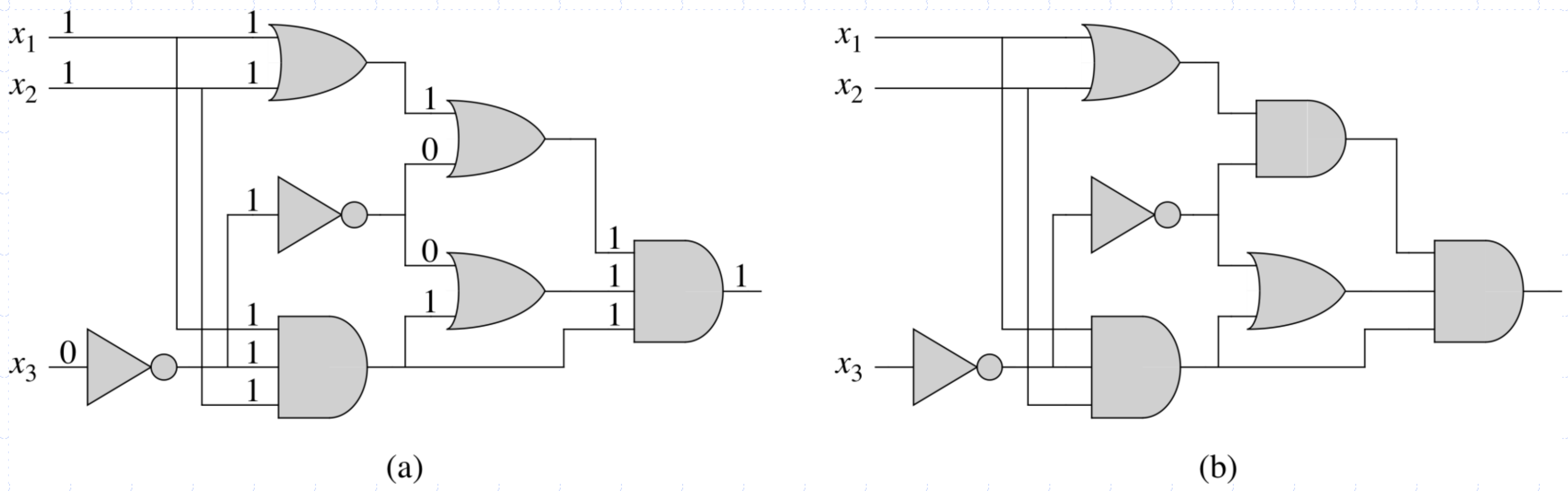
x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

(c)

These gates are connected by wires to each other and to the circuit inputs and a single output. No loops are allowed.

Circuit Satisfiability

- A truth assignment for a BCC is a set of Boolean input values. A truth assignment is satisfying if it causes the output to become 1.



The circuit satisfiability problem is: Given a circuit, is there a satisfying truth assignment for it?

Circuit Satisfiability

We'll call the language form of this problem CIRCUIT-SAT.

Lemma. CIRCUIT-SAT is in NP.

Proof. We merely need a polynomial algorithm to verify CIRCUIT-SAT; this algorithm would take the circuit C (encoded in some standard form) and a certificate, which could be an assignment of boolean values to the wires of C. The algorithm would only need to check each gate to see if the boolean values on its input wires correctly leads to the boolean value on its output wire. If all these are correct, it returns the output wire's boolean value. All this is easily accomplished in time linear in the size of the graph. ■

Representing a Configuration in a Computation

Suppose we have a computation being done on a computer. We will call the state of the computation at any one time a configuration of the computation.

Suppose we are computing membership of x in a language L that is in NP. We could expect the configuration to look like:

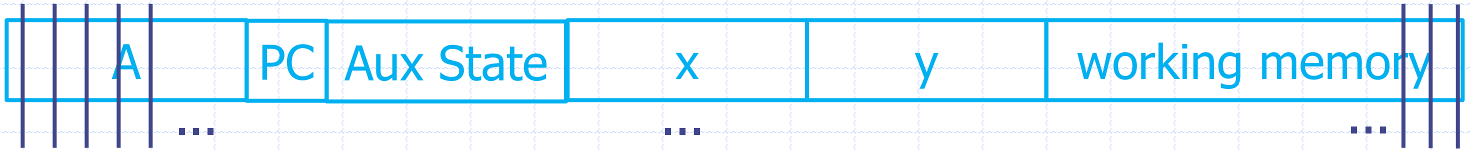
A	PC	Aux State	x	y	working memory

Where PC is the computer's program counter, aux state is auxilliary state information about the computer (register contents and so on), x is the string we're determining membership for, and y is the certificate. Somewhere in working memory is a location for the result.

Representing a Configuration in a Computation

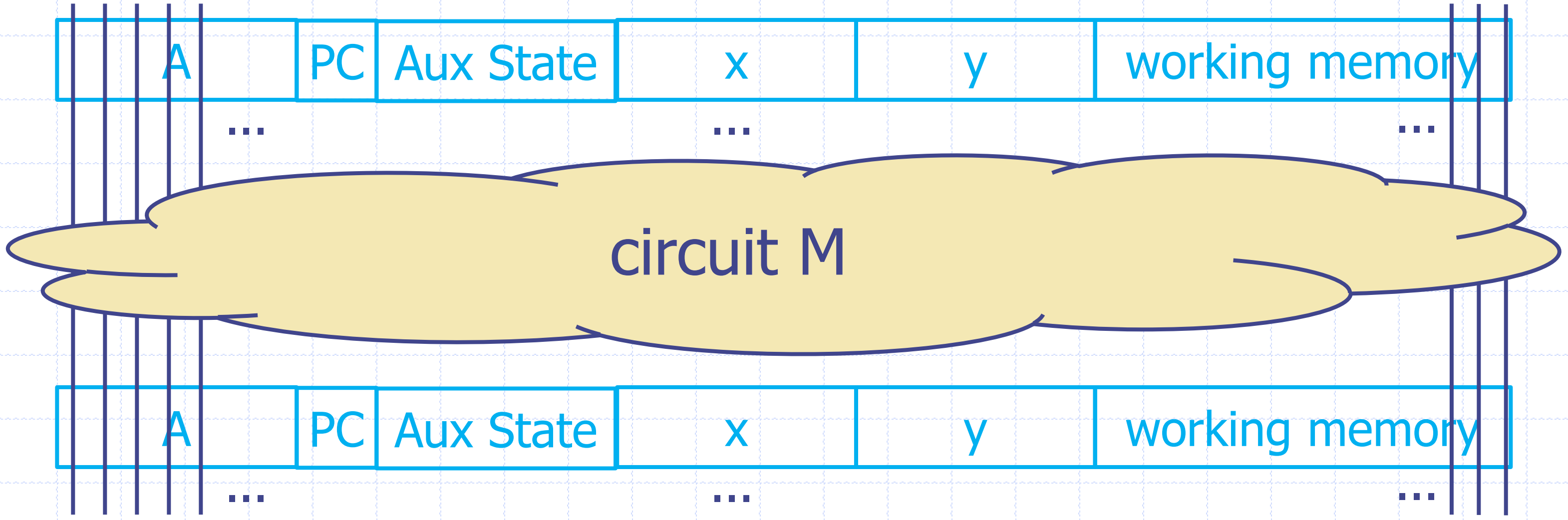
We want to represent a configuration in a BCC instance. Since the configuration is in a computer, it's just a big long sequence of 0's and 1's. How long? Polynomially long in x.

Each bit in the configuration sequence can be represented by the boolean value of one wire in the BCC:



When the computer executes an instruction, we step from one configuration to another:

From One Configuration to Another



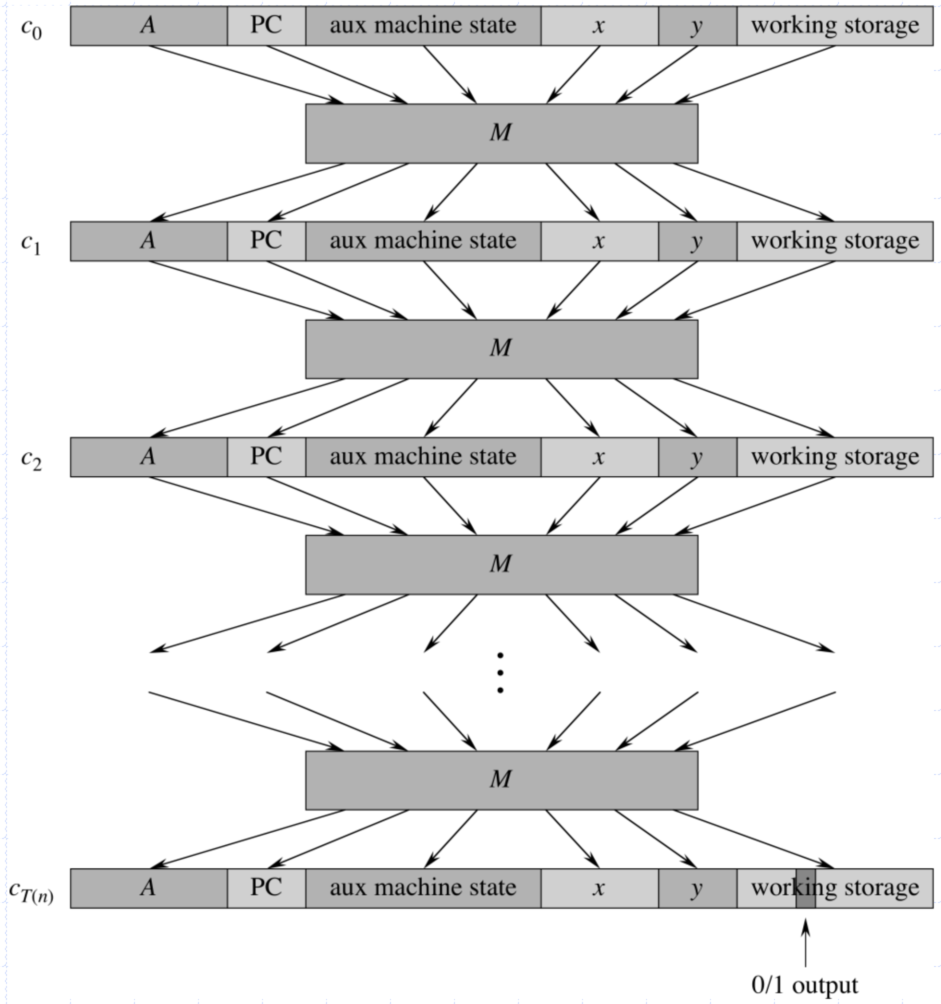
That cloudy circuit can be implemented with a bunch of BCCs...one for each bit in the output configuration. And each BCC is polynomial-sized in x. Thus the whole cloud is polynomial-sized in x.

From One Configuration to Another

But the computation that A performs to validate x must be polynomial in x.

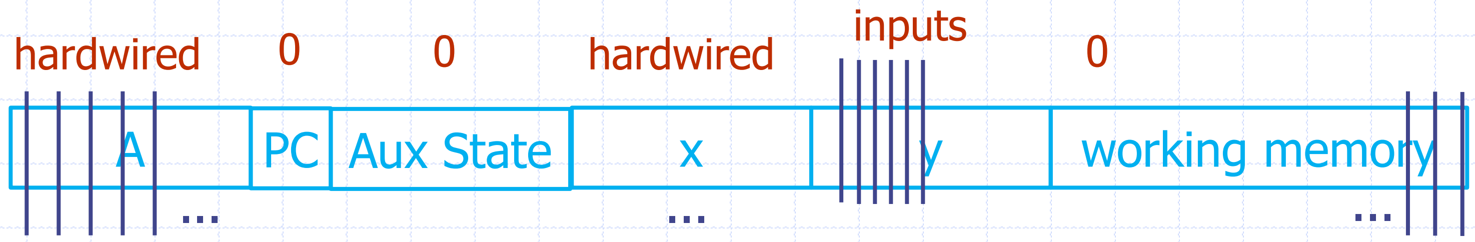
So we get a circuit M' with a polynomial number of copies of M from top to bottom. Since M is polynomial, the whole construction is polynomial.

And the final output of the circuit is simply the location of working storage meant to hold the result.



Starting Configuration

The algorithm A is known and x is known when we construct M', which is a function of x. The size of y is known but y is not known. These unknown bits that comprise y will be the inputs to our BCC M'.



By "hardwired", we mean the bits are each wired to a circuit that generates a 1 or a 0, as appropriate to encode A or x. By "0", we mean all bits are hardwired to 0.

Recap

Given a language L in NP, there is an algorithm A that runs in polynomial time that verifies membership in L in polynomial time, given an instance x and certificate y.

From A, and x, we construct a Boolean Combinational Circuit M' that simulates a computer executing A on inputs x and y. The only inputs of M' are the bits for y.

If A accepts x and y, the computation simulation puts a 1 in the designated output location, and this means there is a satisfying truth assignment for the circuit M'.

Since M' is polynomial in the size of x, we have shown that for any L in NP, L $\leq_P$ CIRCUIT-SAT.

Thus, CIRCUIT-SAT is NP-Complete.

Formula Satisfiability

Formula Satisfiability, or SAT, is the problem:

Instance: A boolean formula Φ
Question: is there a setting of the variables in Φ that causes Φ to be true?

The formula consists of

1. n Boolean variables $x_1, x_2, \ldots, x_n$;
2. m Boolean connectives: any Boolean function with one or two inputs and one output, such as $\wedge$ (AND), $\vee$ (OR), $\neg$(NOT), $\rightarrow$ (IMPLIES), and $\leftrightarrow$ (IFF); and
3. parentheses. WLOG we assume that there are no redundant parentheses, i.e. there is at most one pair of parentheses per Boolean connective.

Formula Satisfiability

An example formula is

$$\Phi = ((x_1 \rightarrow x_2) \vee ((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$

This formula is satisfiable; let $x_1 = 0$, $x_2 = 0$, $x_3 = 1$, $x_4 = 1$:

$$\begin{aligned}\Phi &= ((0 \rightarrow 0) \vee ((\neg 0 \leftrightarrow 1) \vee 1)) \wedge \neg 0\\ &= (1 \vee ((1 \leftrightarrow 1) \vee 1)) \wedge 1\\ &= (1 \vee (1 \vee 1)) \wedge 1\\ &= 1 \wedge 1\\ &= 1\end{aligned}$$

It is easy to encode a Boolean formula Φ in a length that is polynomial in $n+m$.

Formula Satisfiability

Theorem. SAT is NP-Complete.

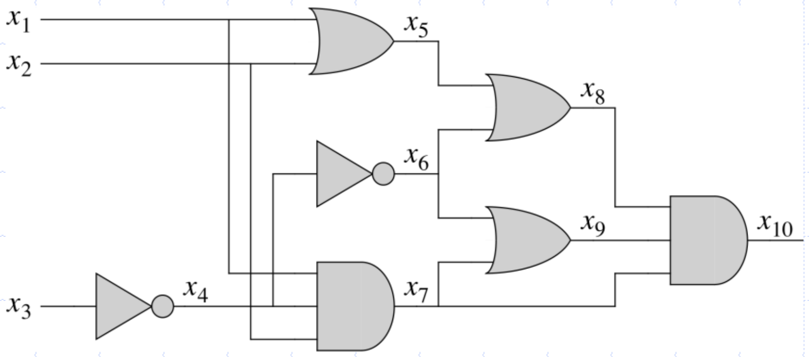
Proof. First we show SAT $\in$ NP. To show this, let C be a certificate consisting of truth assignments for all variables in formula Φ. The verification algorithm must then only go through the formula replacing variables by their assigned Boolean values, and then evaluate the expression, as we did on the previous slide. This can easily be done in polynomial time. If the expression evaluates to 1, the formula is satisfiable. So SAT $\in$ NP.

Now we must show that SAT is NP-Hard. We do this by reducing a known NP-Hard problem to SAT. The only known NP-Hard problem at this point is CIRCUIT-SAT, so we will show CIRCUIT-SAT $\leq_P$ SAT.

So suppose we are given a CIRCUIT-SAT instance I. We transform this in polynomial time to a SAT instance I' where the answer is the same.

Formula Satisfiability

For each wire in instance I, there is a Boolean value along that wire in a hypothetical satisfying assignment of inputs for I. We will let the variables $x_1, x_2, \ldots x_n$ of I' correspond to the n wires of I. We then construct a clause of the formula of I' for each gate of I, and AND all the clauses together with the output wire variable.



$$\begin{aligned}\Phi = x_{10} &\wedge (x_4 \leftrightarrow \neg x_3)\\ &\wedge (x_5 \leftrightarrow (x_1 \vee x_2))\\ &\wedge (x_6 \leftrightarrow \neg x_4)\\ &\wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4))\\ &\wedge (x_8 \leftrightarrow (x_5 \vee x_6))\\ &\wedge (x_9 \leftrightarrow (x_6 \vee x_7))\\ &\wedge (x_{10} \leftrightarrow (x_8 \wedge x_9 \wedge x_7))\end{aligned}$$

Formula Satisfiability

Each gate's clause having to be true enforces that the variable corresponding to the gate's output is the correct value given the gate's inputs. The output variable can be true only if there is some assignment to the circuit input variables that causes the circuit to compute true. Thus, if the formula is satisfied, there is a satisfying input to the circuit.

Similarly, if the circuit has a satisfying input assignment, we can assign each variable to the value on the corresponding wire of the circuit. This must be a satisfying assignment because each clause is satisfied, as can be verified by looking at the corresponding combinational component of the circuit.

Thus, the instance I of CIRCUIT-SAT is a yes-instance iff the instance I' of SAT is a yes-instance. So CIRCUIT-SAT $\leq_P$ SAT, SAT is NP-Hard, and SAT is NP-Complete. ■