3-SAT Is NP-Complete

In the previous lecture, we showed that the propositional satisiability problem for CNF formulas is NP-complete. In this lecture, we will start proving that other problems are NP-complete. We will start with the problem which is the closest to satisfiability for CNF formulas – the *3-SAT* problem, the propositional satisfiability problem for 3-SAT formulas.

First, we need to explain what are 3-SAT formulas. For this, we will have to recall some previously studied definitions.

1 Reminder

Definitions. A *Boolean variable* is a variable that can only take two values: "true" (1) and "false" (0). A *propositional formula* is any formula that is obtained from the Boolean variables by using "and" (&), "or" ($\vee$), and "not" ($\neg$). An example of a propositional formula is $\neg(v_1 \vee (\neg v_2 \,\&\, v_3))$.

A *literal* is a variable or its negation. Examples of literals: v_1, $\neg v_2$. A *clause* (or *disjunction* is an expression of the type $a \vee \ldots \vee b$, where $a, \ldots, b$ are literals. Examples of clauses: $v_1 \vee \neg v_2$, $v_1 \vee \neg v_2 \vee \neg v_3$. Finally, a *CNF-formula* (i.e., a formula in Conjunctive Normal Form) is an expression of the type $C_1 \,\&\, \ldots \,\&\, C_k$, where C_i are clauses. Example:

$$(v_1 \vee \neg v_2) \,\&\, (v_1 \vee v_2 \vee \neg v_3).$$

Terminological comment. CNF formulas are called this way, because when we compute the value of this formula, the last operation is the "and"-operation – and this operation is also known as *conjunction*, from the Latin word which is similar to Spanish conjunto "together".

Propositional satisfiability problem. The propositional satisfiability problem for CNF formulas is the following problem:

- *given:* a CNF formula;
- *find:* the values of the Boolean variables that make this formula true (or return a message that no such values exist).

This problem is known as SAT.

For example, for the above CNF formula, one of the possible solutions is $v_1 =$"false", $v_2 =$"false", and $v_3 =$"false". One can easily check that for these values, the above formula is true.

2 3-CNF and 3-SAT: New Notions

3-CNF. The notion of a 3-CNF formula is easy to describe: it is a CNF formula for which every clause has at most 3 literals. For example, the above formula $(v_1 \vee \neg v_2) \,\&\, (v_1 \vee v_2 \vee \neg v_3)$ is a 3-CNF formula, since both clauses have no more than 3 literals.

On the other hand, a CNF formula

$$(v_1 \vee \neg v_2) \,\&\, (v_1 \vee v_2 \vee \neg v_3 \vee v_5)$$

is *not* a 3-CNF formula, since its second clause has 4 literals: v_1, $\neg v_2$, $\neg v_3$, and v_4.

3-SAT. The propositional satisfiability problem for 3-CNF formulas is the following problem:

- *given:* a 3-CNF formula;
- *find:* the values of the Boolean variables that make this formula true (or return a message that no such values exist).

This problem will be denoted by 3-SAT, for short.

3 How We Will Prove That 3-SAT Is NP-Complete: the Main Idea

We want to prove that the problem 3-SAT is NP-complete. After the complicated proof that SAT is NP-complete, you may be expecting a similarly long proof. Good news: this proof – and the proofs that will come after this one – will be much much simpler. Let us explain why.

For this, let us recall what does NP-complete mean. As you remember, a problem from the class NP is NP-complete if every other problem from the class NP can be reduced to this problem. Clearly, 3-SAT is a problem from the class NP:

- if someone proposes a candidate for a solution, i.e., a set of Boolean values,
- then we can plug them in into the formula and in linear time check whether the resulting value is true or not.

So, all we need to do is to prove that every problem from the class NP can be reduced to 3-SAT.

For this, we can use the already proved result – that SAT is NP-complete, i.e., that every problem from the class NP can be reduced to SAT.

It turns out that to prove that *every* problem from the class NP can be reduced to 3-SAT, it is sufficient to prove that SAT can be reduced to 3-SAT. If we prove this reduction, then every problem from the class NP:

- can be reduced to SAT, and then
- SAT can be reduced to 3-SAT.

Combining these two reductions, we get a reduction of the original problem from the class NP to 3-SAT.

Thus, we will then prove that every problem from the class NP can be reduced to 3-SAT – which means exactly that 3-SAT is NP-complete.

Comment. This argument can be explained on the following similar example.

How can we find the tallest person in the class? We need to compare everyone's height and make sure that the tallest person is taller than everybody else.

But once we have found the tallest person and a new person joins the class, how can we tall that the new person is now the tallest?

- We no longer need to compare the new person's height with everybody else's.
- All need to show is that this person is taller than the current champion.

Same with NP-completeness. NP-complete means that the problem is the most complex among problems from the class NP. So, to prove that the new problem is the most complex, it is enough to show that it is more complex than (or at least as complex as) the current most-complex problem.

Let us now show how to reduce SAT to 3-SAT.

4 How to Reduce SAT to 3-SAT: Idea

SAT is about general CNF formulas, 3-SAT is about formulas where each clause has no more than 3 literals. So, all we have to do is to learn how to reduce long clauses with many literals to clauses with 3 or fewer literals.

To illustrate how this can be done let us first consider a literal with 4 clauses $a \vee b \vee c \vee d$. If we know the values of the literals a, b, c, and d, how will we compute the value of this expression?

- First, we will compute $a \vee b$ – let us denote the result by r_1.
- Then, we will compute $r_1 \vee c \vee d$.

So, the above clause is satisfied if and only if we can satisfy the formula

$$(a \vee b = r_1) \,\&\, (r_1 \vee c \vee d).$$

The second part of this formula is already a clause with 3 literals. The first part $a \vee b = r_1$ – which we will denote by F – can be transformed into the CNF

form the usual way. Let us first find the truth values of this formula F and the truth values of its negation. The results are presented in the following table:

a	b	r_1	$a \vee b$	F	$\neg F$
0	0	0	0	1	0
0	0	1	0	0	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	0	1
1	1	1	1	1	0

If we look at all the rows of this table that correspond to $\neg F = 1 =$"true", we will see that the formula $\neg F$ is true only in the following four cases:

- in the case when $a = 0$, $b = 0$, and $r_1 = 1$, i.e., in the case when we have

$$\neg a \,\&\, \neg b \,\&\, r_1;$$

- in the case when $a = 0$, $b = 1$, and $r_1 = 0$, i.e., in the case when we have

$$\neg a \,\&\, b \,\&\, \neg r_1;$$

- in the case when $a = 1$, $b = 0$, and $r_1 = 0$, i.e., in the case when we have

$$a \,\&\, \neg b \,\&\, \neg r_1;$$

- in the case when $a = 1$, $b = 1$, and $r_1 = 0$, i.e., in the case when we have

$$a \,\&\, b \,\&\, \neg r_1.$$

Thus,

$$\neg F \equiv (\neg a \,\&\, \neg b \,\&\, r_1) \vee (\neg a \,\&\, b \,\&\, \neg r_1) \vee (a \,\&\, \neg b \,\&\, \neg r_1) \vee (a \,\&\, b \,\&\, \neg r_1).$$

To get a CNF formula for F, we apply negation to both sides of this equivalence. According to de Morgan laws, when we apply negation, "and" gets transformed into "or", "or" into "and", and each literal is transformed into its negation. So, we get

$$F \equiv (a \vee b \vee \neg r_1) \,\&\, (a \vee \neg b \vee r_1) \,\&\, (\neg a \vee b \vee r_1) \,\&\, (\neg a \vee \neg b \vee r_1).$$

Substituting this expression for F (i.e., for $a \vee b = r_1$) into the expression

$$(a \vee b = r_1) \,\&\, (r_1 \vee c \vee d),$$

we get the following 3-CNF equivalent for the original clause:

$$(a \vee b \vee \neg r_1) \,\&\, (a \vee \neg b \vee r_1) \,\&\, (\neg a \vee b \vee r_1) \,\&\, (\neg a \vee \neg b \vee r_1) \,\&\, (r_1 \vee c \vee d).$$

If we have a clause with 5 literals $a \vee b \vee c \vee d \vee e$, then we first compute $r_1 = a \vee b$, then $r_2 = r_1 \vee c$, and finally $r_2 \vee d \vee e$. So, the satisfiability of the original clause is equivalent to satisfiability of the formula

$$(r_1 = a \vee b) \,\&\, (r_2 = r_1 \vee c) \,\&\, (r_2 \vee d \vee e).$$

The first two parts of this formula can be transformed into CNF – we already know how to do it. So, as a result, we get a 3-CNF formula.

Now, we we are ready to describe the general algorithm.

5 Reducing CNF to 3-CNF: General Algorithm

If we have a CNF formula in which some clauses $a_1 \vee \ldots \vee a_k$ have $k > 3$ literals, then for each such clause, we do the following:

- first, we replace this clause with a formula

$$(a_1 \vee a_2 = r_1) \,\&\, (r_1 \vee a_3 = r_2) \,\&\, \ldots \,\&\, (r_{k-4} \vee a_{k-2} = r_{k-3}) \,\&\, (r_{k-3} \vee a_{k-1} \vee a_k);$$

- then, we replace each expression $a \vee b = c$ by its above-found CNF form

$$(a \vee b \vee \neg c) \,\&\, (a \vee \neg b \vee c) \,\&\, (\neg a \vee b \vee c) \,\&\, (\neg a \vee \neg b \vee c).$$

Comment. This reduction described the algorithm U_1 – that, based on a CNF formula x, builds the corresponding 3-CNF formula $x' = U_1(x)$. One we have a satisfying vector y' for the 3-CNF formula $x' = U_1(x)$, getting the satisfying vector $y = U_2(y')$ is easy: all we have to do is ignore the values of all the auxiliary variables that we introduced (r_1, r_2, $\ldots$) and only keep the truth values of the original Boolean variables.

The algorithm U_3 that transforms a satisfying vector y for the original CNF formula into a satisfying vector $y' = U_3(y)$ of the corresponding 3-CNF formula is slightly more complicated, but still easy: we keep the same truth values for all the original Boolean variables, and we compute the values of all the auxiliary variables from their meaning. For example:

- we can find the truth value of r_1 as $r_1 = a_1 \vee a_2$;
- then, if needed, we can find the truth value of r_2 as $r_2 = r_1 \vee a_3$, etc.

6 Example

Example. Let us consider the above example of a CNF formula

$$(v_1 \vee \neg v_2) \,\&\, (v_1 \vee v_2 \vee \neg v_3 \vee v_5).$$

In this formula, the first clause is OK, but the second has length $4 > 3$, so this formula is not in 3-CNF form.

So, according to the general algorithm, we first replace the second clause with the expression $(v_1 \vee v_2 = r_1) \,\&\, (r_1 \vee \neg v_3 \vee v_5)$. Then, we replace the part $v_1 \vee v_2 = r_1$ with its CNF form

$$(v_1 \vee v_2 \vee \neg r_1) \,\&\, (v_1 \vee \neg v_2 \vee r_1) \,\&\, (\neg v_1 \vee v_2 \vee r_1) \,\&\, (\neg v_1 \vee \neg v_2 \vee r_1).$$

As a result, we get the following 3-CNF formula

$$(v_1 \vee \neg v_2) \,\&\,$$

$$(v_1 \vee v_2 \vee \neg r_1) \,\&\, (v_1 \vee \neg v_2 \vee r_1) \,\&\, (\neg v_1 \vee v_2 \vee r_1) \,\&\, (\neg v_1 \vee \neg v_2 \vee r_1) \,\&\,$$

$$(r_1 \vee \neg v_3 \vee v_5).$$

Try it yourself. Try this algorithm on the example of the following CNF formula: $(p \vee \neg q) \,\&\, (\neg p \vee \neg q \vee \neg r \vee \neg s)$.

Homework. Now, you are hopefully ready to do the corresponding homework.