Lecture 2

This lecture is devoted to the proof that SAT, a decision problem from logics, is **NP**-complete. This result is due to Cook [Coo71]. Cook introduced the notion of reductions and **NP**-completeness, recognized the relevance of the notion of **NP**-completeness, and proved the **NP**-completeness of SAT and of a few other problems. Independently, Levin [Lev73] also developed the notions of reductions and **NP**-completeness, and proved the **NP**-completeness of some natural problems. The **NP**-completeness of SAT is often referred to as *Cook's theorem* or *Cook-Levin's theorem.* Karp [Kar72] proved the **NP**-completeness of several problems, including several optimization problems. Karp's work pointed out the potential of **NP**-completenss as a way of explaining the hardness of an extreme variety of problems. This promise was quickly fulfilled, and **NP**-completeness results came by the hundreds during the 70s.

1 The Problem SAT

In SAT (that stands for *CNF-satisfiability*) we are given Boolean variables $x_1, x_2, \ldots, x_n$ and a Boolean formula ϕ involving such variables; the formula is given in a particular format called *conjunctive normal form*, that we will explain in a moment. The question is whether there is a way to assing Boolean (True / False) values to the variables so that the formula is satisfied.

To complete the description of the problem we need to explain what is a Boolean formula in conjunctive normal form. First of all, Boolean formulas are constructed starting from variables and applying the operators $\vee$ (that stands for OR), $\wedge$ (that stands for AND) and $\neg$ (that stands for NOT).

The operators work in the way that one expects: $\neg x$ is True if and only if x is False; $x \wedge y$ is True if and only if both x and y are True; $x \vee y$ is True if and only at least one of x or y is True.

So, for example, the expression $\neg x \wedge (x \vee y)$ can be satisfied by setting x to False and y to True, while the expression $x \wedge (\neg x \vee y) \wedge \neg y$ is impossible to satisfy.

A *literal* is a variable or the negation of a variable, so for example $\neg x_7$ is a literal and so is x_3. A *clause* is formed by taking one or more literals and connecting them with a OR, so for example $(x_2 \vee \neg x_4 \vee x_5)$ is a clause, and so is (x_3). A *formula in conjunctive normal form* is the AND of clauses. For example

$$(x_3 \vee \neg x_4) \wedge (x_1) \wedge (\neg x_3 \vee x_2)$$

is a formula in conjunctive normal form (from now on, we will just say "CNF formula" or "formula"). Note that the above formula is satisfiable, and, for example, it is satisfied by setting all the variables to True (there are also other possible assignments of values to the variables that would satisfy the formula).

On the other hand, the formula

$$x \wedge (\neg x \vee y) \wedge \neg y$$

is not satisfiable, as it has already been observed.

2 Intuition for the Reduction

From now on, it will be convenient to think of Boolean variables as taking values in $\{0,1\}$, with the convention that 0 stands for FALSE and 1 stands for TRUE.

It may seem that CNF formulas can only specify very simple Boolean relations, however the following result holds.

LEMMA 1
For every function $f : \{0,1\}^k \to \{0,1\}$ there is a CNF formula ϕ over variables $x_1, \ldots, x_k$ such that an assignment $b_1, \ldots, b_k$ to $x_1, \ldots, x_k$ satisfies ϕ if and only $f(b_1, \ldots, b_k) = 1$. Furthermore, the size of ϕ is at most $k2^k$.

So it is possible to represent every function using a CNF formula, although the formula promised by the above lemma could be exponentially big in the number of variables. If we could guarantee the formula to be small for all efficiently computable f, then the **NP**-completeness of SAT would be quite easy to establish.

Let L be an **NP** decision problem. Then, by definition, there is an algorithm V_L and polynomials T and p such that for every x, we have that $x \in L$ if and only if there is a y, $|y| \leq p(|x|)$ such that $V_L(x,y)$ accepts; furthermore $V_L(x,y)$ runs at most in time $T(|x|+|y|)$. Now, fix an x, call $l = p(|x|)$, and consider the function $f_x : \{0,1\}^l \to \{0,1\}$ such that $f_x(y) = 1$ if and only if $V_L(x,y)$ accepts. Suppose we could produce a CNF formula ϕ_x over l variables such that ϕ_x is satisfied only by assignments that correspond to a y such that $f_x(y) = 1$. Then ϕ_x is satisfiable if and only if $x \in L$, and the transformation mapping x in ϕ_x would be a reduction from L to SAT. Since L was arbitrary, this argument would prove the **NP**-hardness of SAT.

Unfortunately, the function f_x could be such that there is no equivalent polynomial-size CNF formula, so it may be impossible to implement this strategy in polynomial time. In fact, even very simple functions have no short CNF formula representation, and the above strategy cannot possibly work, as shown by the next example.

LEMMA 2
Let ϕ be a formula over k variables, such that ϕ is satisfied precisely by those assignments where an odd number of variables takes value 1. Then ϕ has at least 2^{k-1} clauses.

3 The Reduction

Fortunately, there is a way to implement the general idea given in the previous section; unfortunately, there is some technical work to do.

Let L be an **NP** decision problem, and let $V(\cdot,\cdot)$ be the algorithm and T and p be the polynomials as above. We assume that V is implemented as a single-tape Turing machine M with semi-infinite tape.

Let us also fix an input x, and let $l = p(|x|)$ and $t = T(|x| + l)$.

Our goal is to construct, in time polynomial in $|x|$, a CNF formula ϕ_x such that ϕ_x is satisfiable if and only if $x \in L$, that is, if and only there is a y, $|y| \leq l$ such that $M(x, y)$ accepts within t steps.

We will construct ϕ_x so that its variables not only represent a possible choice of y, but also give a complete representation of the whole computation of $M(x, y)$. Let Q be the set of states of M, and let Σ be the alphabet of M. Since we want to represent computations of M that take time at most t, we know that only the first t entries of the tape are relevant for the description. The global state (also called a *configuration*) of the machine can be described by giving the position of the head on the tape (in one of the possible t positions), the current state of the machine (one of the $|Q|$ possible ones), and giving the content of the relevant part of the tape (a string in Σ^t). We will choose a more redundant representation, that will be easier to deal with. For each cell of the tape, we specify a pair (q, a), where $q \in Q \cup \{\Box\}$ is the state of the machine, if the head is on that particular cell, or the special symbol $\Box$ otherwise, and a is the content of the cell. So the configuration is represented by a string in $(\Sigma \times (Q \cup \{\Box\}))^t$. A computation of length t can be represented by a sequence of $t + 1$ such configurations. Such a $t \times (t + 1)$ table of elements of $\Sigma \times (Q \cup \{\Box\})$ will be called the *tableau* of a computation.

The variables of the formula ϕ_x are meant to represent a tableau. For every $i = 0, \ldots, t$, $j = 1, \ldots, t$, $q \in Q \cup \{\Box\}$, $a \in \Sigma$, we have a variable $z_{i,j,q,a}$. The intended meaning of the variable is that the variable is true if and only if the tableau contains the pair (q, a) in the entry corresponding to time i and position j on the tape. Overall, we have $t \times (t + 1) \times |\Sigma| \times (|Q| + 1)$ variables.

The clauses of ϕ_x, that we will describe in a moment, enforce the property that every assignment to $z_{i,j,q,a}$ that satisfies ϕ_x encodes the tableau of a computation $M(x, y)$ where $|y| \leq l$, and M accepts at the end.

First of all, we need to enforce that the variables are giving a consistent description of the tableau, that is, that for each fixed i and j, there is exactly one pair (q, a) such that $z_{i,j,q,a}$ is true. This is enforced by the clauses

$$(\bigvee_{q,a} z_{i,j,q,a}) \text{ for all } i = 0, \ldots, t,\ j = 1, \ldots, t$$

and the clauses

$$(\neg z_{i,j,q,a} \vee \neg z_{i,j,q',a'}) \text{ for all } i = 0, \ldots, t,\ j = 1, \ldots, t,\ (q, a) \neq (q', a')$$

We also want to make sure that, in every line, exactly one cell contains a pair (q, a) where $q \neq \Box$. We will use the clauses

$$(\bigvee_{q \neq \Box, a, j} z_{i,j,q,a}) \text{ for all } i$$

and the clauses

$$(\neg z_{i,j,q,a} \vee \neg z_{i,j',q',a'}) \text{ for all } i, j \neq j', q, q' \in Q, a, a' \in \Sigma$$

Next, we want to enforce that the first line corresponds the a starting configuration where the tape contains (x, y) for some y of length l. Let q_0 be the initial state, let $n = |x|$, and let $x = (x_1, \ldots, x_n)$. We will use the clauses

$$z_{0,1,q_0,x_1}$$

$$z_{0,j,\square,x_i} \text{ for all } j = 2, \ldots, n$$

$$(z_{0,j,\square,0} \vee z_{0,j,\square,1}) \text{ for all } j = n+1, \ldots, n+l$$

$$z_{0,j,\square,\square} \text{ for all } j = n+l+1, \ldots, t$$

Then, we want to enforce that each line of the tableau is consistent with the previous line. Note that the content of the entry (i, j) of the tableau only depends on the entries $(i-1, j-1)$, $(i-1, j)$ and $(i-1, j+1)$ of the tableau (for $i \geq 2$). We can then use Lemma 1 to claim that for each i, j, $i \geq 2$, there is a CNF formula $\phi_{i,j}$ over the variables of the form $z_{i,j,q,a}$, $z_{i-1,j-1,q,a}$, $z_{i-1,j,q,a}$ and $z_{i-1,j-1,q,a}$ that is satisfied if and only if the variables encode a transition that is consistent with behaviour of M. Each such formula $\phi_{i,j}$ relates to $4 \times (|Q| + 1) \times |\Sigma| = O(1)$ variables, and so it is of size $O(1)$. We can construct all those formulas, and then take their AND, together with all the clauses described above.

Finally, we add the clause $\vee_{j,a} z_{t,j,q_A,a}$, where q_A is the accepting state of M.

The resulting formula is ϕ_x. Looking back at the description of the clauses in ϕ_x, the construction of ϕ_x can be done in time polynomial in $|x|$.

We claim that ϕ_x is satisfiable if and only if $x \in L$; this proves that L is reducible to SAT.

Suppose $x \in L$, then there is y, $|y| \leq l$ such that $M(x, y)$ accepts in at most t steps. Write down the tableau of the computation of $M(x, y)$, then compute the corresponding assignment to variables $z_{i,j,q,a}$. One can verify that such an assignment satisfies all the clauses described above.

Suppose we have an assignment of values $b_{i,j,q,a}$ to variables $z_{i,j,q,a}$ that satisfies ϕ_x. Construct the tableau corresponding to the values $b_{i,j,q,a}$. Since all clauses of ϕ_x are satisfied, the tableau describes a computation $M(x, y)$, for some y of length l, such that $M(x, y)$ accepts in at most t steps. Then it must be the case that $x \in L$.

Since L was arbitrary, we deduce that SAT is **NP**-hard. It's easy to see that SAT is in **NP**, so we proved that SAT is **NP**-complete.

4 The NP-Completeness of 3SAT

3SAT is the restriction of SAT to formulas where every clause is the OR of precisely 3 literals. A CNF formula where every clause contains exactly 3 literals is also called a 3CNF formula.

We give a reduction from SAT to 3SAT, thereby showing that 3SAT is NP-complete.

Let ϕ be a CNF formula, we want to create a new formula ϕ_3 such that ϕ is satisfiable if and only ϕ_3 is satisfiable; furthermore ϕ_3 contains only clauses with exactly 3 literals.

Note that it is not possible to construct ϕ_3 so that it has the same set of variables as ϕ, and is satisfied precisely by the same assignments. Consider for example the case where ϕ contains only the clause $(x_1 \vee x_2 \vee x_3 \vee x_4)$. There is no 3CNF formula over variables x_1, x_2, x_3, x_4 that has the same set of satisfying assignments as ϕ.

Starting from an arbitrary CNF formula ϕ, we will construct ϕ_3 by substitution each clause in ϕ that contains 1, 2, 4 or more literals by a small 3CNF formula, involving new variables. Each substitution will preserve satisfiability.

A clause of the form (x) is replaced by

$$(x \vee y_1 \vee y_2) \wedge (x \vee \neg y_1 \vee y_2) \wedge (x \vee y_1 \vee \neg y_2) \wedge (x \vee \neg y_1 \vee \neg y_2)$$

where y_1, y_2 are new variables.

A clause of the form $(x_1 \vee x_2)$ is replaces by

$$(x_1 \vee x_2 \vee y) \wedge (x_1 \vee x_2 \vee \neg y)$$

where y is a new variable.

A clause of the form $(x_1 \vee \cdots \vee x_k$, $k \geq 4$ is replaced by

$$(x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee x_3 \vee y_2) \wedge \cdots \wedge (\neg y_{k-3} \vee x_{k-1} \vee x_k)$$

where $y_1, \ldots, y_{k-3}$ are new variables.

One should check that each of these substitutions do indeed preserve satisfiability, and so that the combination of all these substitutions give a reduction that produces a formula ϕ_3 that is satisfiable if and only if ϕ is satisfiable.

References

[Coo71] S.A. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[Kar72] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[Lev73] L. A. Levin. Universal search problems. *Problemi Peredachi Informatsii*, 9:265–266, 1973.