5.4 NP-completeness

We say that a language $\mathcal{L}_1 \subset \Sigma_1^*$ is **polynomially reducible** to a language $\mathcal{L}_2 \subset \Sigma_2^*$ if there is a function $f : \Sigma_1^* \to \Sigma_2^*$ computable in polynomial time such that for all words $x \in \Sigma_1^*$ we have

$$x \in \mathcal{L}_1 \Leftrightarrow x \in \mathcal{L}_2.$$

It is easy to verify from the definition that this relation is transitive:

(5.1) **Proposition** *If $\mathcal{L}_1$ is polynomially reducible to $\mathcal{L}_2$ and $\mathcal{L}_2$ is polynomially reducible to $\mathcal{L}_3$ then $\mathcal{L}_1$ is polynomially reducible to $\mathcal{L}_3$.*

The membership of a language in P can also be expressed by saying that it is polynomially reducible to the language $\{0, 1\}$.

(5.2) **Proposition** *If a language is in* P *then every language is in* P *that is polynomially reducible to it. If a language is in* NP *then every language is in* NP *that it polynomially reducible to it.*

We call a language NP-**complete** if it belongs to NP and every language in NP is polynomially reducible to it. These are thus the "hardest" languages in NP. The word "completeness" suggests that the solution of the decision problem of a complete language contains, in some sense, the solution to the decision problem of all other NP languages. If we could show about even a single NP-complete language that it is in P then P = NP would follow. The following observation is also obvious.

(5.3) **Proposition** *If an* NP*-complete language $\mathcal{L}_1$ is polynomially reducible to a language $\mathcal{L}_2$ in* NP *then $\mathcal{L}_2$ is also* NP*-complete.*

It is not obvious at all that NP-complete languages exist. Our first goal is to give an NP-complete language; later (by polynomial reduction, using 5.3) we will prove the NP-completeness of many other problems.

A Boolean polynomial is called **satisfiable** if the Boolean function defined by it is not identically 0.

(5.4) SATISFIABILITY PROBLEM For a given Boolean polynomial f, decide whether it is satisfiable. We consider the problem, in general, in the case when the Boolean polynomial is a conjunctive normal form. $\diamondsuit$

We can consider each conjunctive normal form as a word over the alphabet consisting of the symbols "x", "0", "1", "+", "$\neg$", "$\wedge$" and "$\vee$" (we write the indices of the variables in binary number system, e.g. $x_6 = x110$). Let SAT denote the language formed from the satisfiable conjunctive normal forms.

The following theorem is one of the central results in complexity theory.

(5.5) **Theorem Cook–Levin Theorem.** *The language* SAT *is* NP*-complete.*

Proof Let $\mathcal{L}$ be an arbitrary language in NP. Then there is a non-deterministic Turing machine $T = \langle k, \Sigma, \Gamma, \Phi \rangle$ and there are integers $c, c_1 > 0$ such that T recognizes $\mathcal{L}$ in time $c_1 \cdot n^c$. We can assume $k = 1$. Let us consider an arbitrary word $h_1 \cdots h_n \in \Sigma^*$. Let $N = \lceil c_1 \cdot n^c \rceil$. Let us introduce the following variables:

$$\begin{aligned} x[n,g] \quad & (0 \le n \le N,\ g \in \Gamma), \\ y[n,p] \quad & (0 \le n \le N,\ -N \le p \le N), \\ z[n,p,h] \quad & (0 \le n \le N,\ -N \le p \le N,\ h \in \Sigma). \end{aligned}$$

If a legal computation of the machine T is given then let us assign to these variables the following values: $x[n,g]$ is true if after the n-th step, the control unit is in state g; $y[n,p]$ is true if after the n-th step, the head is on the p-th tape cell; $z[n,p,h]$ is true if after the n-the step, the p-th tape cell contains symbol h. The variables x, y, z obviously determine the computation of the Turing machine.

However, not each possible system of values assigned to the variables will correspond to a computation of the Turing machine. One can easily write up logical relations among the variables that, when taken together, express the fact that this is a legal computation accepting $h_1 \cdots h_n$. We must require that the control unit be in some state in each step:

$$\bigvee_{g \in \Gamma} x[n,g] \qquad (0 \le n \le N);$$

and it should not be in two states:

$$\neg x[n,g] \vee \neg x[n,g'] \qquad (g, g' \in \Gamma,\ 0 \le n \le N).$$

We can require, similarly, that the head should be only in one position in each step and there should be one and only one symbol in each tape cell. We write that initially, the machine is in state START and at the end of the computation, in state STOP, and the head starts from cell 0:

$$x[0, \mathrm{START}] = 1, \qquad x[N, \mathrm{STOP}] = 1, \qquad y[0,0] = 1;$$

and, similarly, that the tape contains initially the input $h_1 \cdots h_n$ and finally the symbol 1 on cell 0:

$$\begin{aligned} z[0, i-1, h_i] &= 1 \qquad (1 \le i \le n) \\ z[0, i-1, *] &= 1 \qquad (i < 0 \text{ or } i > n) \\ z[N, 0, 1] &= 1. \end{aligned}$$

We must further express the computation rules of the machine, i.e., that for all $g, g' \in \Gamma$, $h, h' \in \Sigma$, $\varepsilon \in \{-1, 0, 1\}$ and $-N \le p \le N$ we have

$$(x[n, g] \wedge y[n, p] \wedge z[n, p, h]) \Rightarrow \neg(x[n+1, g'] \wedge y[n+1, p+\varepsilon] \wedge z[n+1, p, h'])$$

and that where there is no head the tape content does not change:

$$\neg y[n, p] \Rightarrow (z[n, p, h] \Leftrightarrow z[n+1, p, h]).$$

For the sake of clarity, the the last two formulas are not in conjunctive normal form but it is easy to bring them to such form. Joining all these relations by the sign "$\wedge$" we get a conjunctive normal form that is satisfiable if and only if the Turing machine T has a computation of at most N steps accepting $h_1 \cdots h_n$. It easy to verify that for given $h_1, \ldots, h_n$, the described construction of a formula can be carried out in polynomial time. ■

It will be useful to prove the NP-completeness of some special cases of the satisfiability problem. A conjunctive normal form is called a k-**form** if in each of its components, at most k literals occur. Let k-SAT denote the language made up by the satisfiable k-forms. Let further SAT-k denote the language consisting of those satisfiable conjunctive normal forms in which each variable occurs in at most k elementary disjunctions.

(5.6) **Theorem** *The language k-SAT is NP-complete.*

Proof Let B be a Boolean circuit with inputs $x_1, \ldots, x_n$ (a conjunctive normal form is a special case of this). We will find a 3-normal form that is satisfiable if and only if the function computed by B is not identically 0. Let us introduce a new variable y_i for each node i of the circuit. The meaning of these variables is that in a satisfying assignment, these are the values computed by the corresponding nodes. Let us write up all the restrictions for y_i. For each input node i, with node variable y_i and input variable x_i we write

$$y_i \Leftrightarrow x_i \qquad (1 \le i \le n).$$

If y_i is the variable for an $\wedge$ node with inputs y_j and y_k then we write

$$y_i \equiv y_j \wedge y_k.$$

If y_i is the variable for a $\vee$ node with inputs y_j and y_k then we write

$$y_i \equiv y_j \vee y_k.$$

If y_i is the variable for a $\neg$ node with input y_j then we write

$$y_i \equiv \neg y_j.$$

Finally, if y_i is the output node then we add the clause

$$y_i.$$

Each of these statements involves only three variables and is therefore expressible as a 3-normal form. The conjunction of all these is satisfiable if and only if B is satisfiable. ∎

it is natural to wonder at this point why have we considered just the 3-satisfiability problem. The problems 4-SAT, 5-SAT, etc. are harder than 3-SAT therefore these are, of course, also NP-complete. The theorem below shows, on the other hand, that the problem 2-SAT is already not NP-complete (at least if P $\neq$ NP). (This illustrates the fact that often a little modification of the conditions of a problem leads from a polynomially solvable problem to an NP-complete one.)

(5.7) **Theorem** *The language 2-*SAT *is in* P.

Proof Let B be a 2-normal form on the variables $x_1, \ldots, x_n$. Let us use the convention that the variables x_i are also written as x_i^1 and the negated variables $\overline{x}_i$ are also written as new symbols x_i^0. Let us construct a directed graph G on the set $V(G) = \{x_1, \ldots, x_n, \overline{x}_1, \ldots, \overline{x}_n\}$ in the following way: we connect node x_i^ε to node x_j^δ if $x_i^{1-\varepsilon} \vee x_j^\delta$ is an elementary disjunction in B. (This disjunction is equivalent to $x_i^\varepsilon \Rightarrow x_j^\delta$.) Let us notice that then in this graph, there is also an edge from $x_j^{1-\delta}$ to $x_i^{1-\varepsilon}$. In this directed graph, let us consider the **strongly connected components**; these are the classes of nodes obtained when we group two nodes in one class whenever there is a directed path between them.

(5.8) **Lemma** *The formula B is satisfiable if and only if none of the strongly connected components of G contains both a variable and its negation.*

The theorem follows from this lemma since it is easy to find in polynomial time the strongly connected components of a directed graph. ∎

Proof of Lemma 5.8 Let us note first that if an assignment of values satisfies formula B and x_i^ε is "true" in this assignment then every x_j^δ is "true" to which an edge leads from x_i^ε: otherwise, the elementary disjunction $x_i^{1-\varepsilon} \vee x_j^\delta$ would not be satisfied. It follows from this that the nodes of a strongly connected component are either all "true" or none of them. But then, a variable and its negation cannot simultaneously be present in a component.

Conversely, let us assume that no strongly connected component contains both a variable and its negation. Consider a variable x_i. According to the condition, there cannot be directed paths in both directions between x_i^0 and x_i^1. Let us assume there is no such directed path in either direction. Let us then draw a new edge from x_i^1 to x_i^0. This will not violate our assumption that no connected component contains both a node and its negation. If namely such a connected components should arise then it would contain the new edge, but then both x_i^1 and x_i^0 would belong to this component and therefore there would be a path from x_i^0 to x_i^1. But then this path would also be in the original graph, which is impossible.

Repeating this procedure, we can draw in new edges (moreover, always from a variable to its negation) in such a way that in the obtained graph, between each variable and its negation, there will be a directed path in exactly one direction. Let now be $x_i = 1$ if a directed path leads from x_i^0 to x_i^1 and 0 if not. We claim that this assignment satisfies all disjunctions. Let us namely consider an elementary disjunction, say, $x_i \vee x_j$. If both of its members were false then—according to the definition—there were a directed path from x_i^1 to x_i^0 and from x_j^1 to x_j^0. Further, according to the definition of the graph, there is an edge from x_i^0 to x_j^1 and from x_j^0 to x_i^1. But then, x_i^0 and x_i^1 are in a strongly connected components, which is a contradiction. ∎

(5.9) **Theorem** *The language* SAT-3 *is* NP-*complete.*

Proof Let B be a Boolean formula of the variables $x_1, \dots, x_n$. For each variable x_j, replace the i-th occurrence of x_j in B, with new variable y_j^i: let the new formula be B'. For each j, assuming there are m occurrences of x_j in B, form the conjunction

$$C_j = (y_j^1 \Rightarrow y_j^2) \wedge (y_j^2 \Rightarrow y_j^3) \wedge \cdots \wedge (y_j^m \Rightarrow y_j^1).$$

(Of course, $y_j^1 \Rightarrow y_j^2 = \neg y_j^1 \vee y_j^2$.) The formula $B' \wedge C_1 \wedge \cdots \wedge C_n$ contains at most 3 occurrences of each variable, is a conjunctive normal form if B is, and is satisfiable obviously if and only if B is. ∎