§12. Cook's theorem. Since a deterministic TM is a special case of a nondeterministic TM it is clear that $P \subseteq NP$. However there is no reason to believe that the classes P and NP are *equal*, and indeed it is widely conjectured that $P \subset NP$.[28] Note that if the conjecture is true, then the class NP contains *some* computationally intractable languages.

Using polynomial-time reductions it is possible to identify a certain class of languages which are, computationally, the 'hardest' in NP. A language L is said to be *NP-hard* if every language in NP is polynomial-time reducible to L. A language L is *NP-complete* if $L \in$ NP and L is NP-hard. The significance of NP-completeness is made clear in the following result:

Corollary 12.1 *Let L be any NP-hard language. L cannot be in P unless we have* $P = NP$.

proof. Suppose $L \in$ P, and let L' be an arbitrary language in NP. Since L is NP-hard, L' must be polynomial-time reducible to L. Therefore, by Theorem 11.1, $L' \in$ P. But L' was chosen arbitrarily from NP, and hence $P = NP$. □

It follows from Corollary 12.1 that the NP-complete languages are either all in P (i.e, tractable) or all outside P (i.e., intractable). The fact that no one has so far found a polynomial-time algorithm for recognizing any NP-complete language provides strong empirical evidence in support of the conjecture that $P \subset NP$. Of course it is also true that nobody has managed to prove that the inclusion really is strict and so it is perfectly possible that $P = NP$. At the moment all we can say with certainty is that we don't know!

The criterion for a language L to be NP-complete is a strong one: *every* language in NP must be polynomial-time reducible to L. Thus it comes as a surprise that NP-complete languages should occur naturally. Indeed the existence of such languages was unsuspected much before 1971, when Stephen Cook demonstrated that Sat is NP-complete. (The result was discovered independently by Leonid Levin in the former Soviet Union, but several years were to pass before his work became known in the West.)

Theorem 12.1 Sat *is NP-complete.*

proof. It was demonstrated, in Note 11, that Sat $\in$ NP. To complete the proof we must show that Sat is NP-hard, i.e., that every language in NP is polynomial-time reducible to Sat.

Suppose L is any language in NP. Let $M = (Q, \Gamma, \Sigma, \flat, q_I, q_F, \delta)$ be a $p(n)$ time-bounded nondeterministic TM recognizing L, where p is some polynomial.

[28] Note that the symbol $\subset$ denotes *strict* containment.

column number

time step

	0	1	2	3	$\cdots$	$n+1$	$n+2$	$\cdots$		$\cdots$	$m+2$	$m+3$
0	#	q_I	x_0	x_1	$\cdots$	x_{n-1}	$\not{b}$	$\cdots$		$\cdots$	$\not{b}$	#
1	#											#
2	$\vdots$											$\vdots$
	$\vdots$											$\vdots$
m	#								q_F			#

Figure 6: A tableau for M on input x.

We show how to construct, given any input $x \in \Sigma^n$ for M, a CNF formula $\phi_M(x)$ such that

$$M \text{ accepts } x \iff \phi_M(x) \text{ is satisfiable.}$$

The mapping from x to $\phi_M(x)$ is computable in time polynomial in the length of x, and hence is a polynomial-time reduction from L to SAT. Since L was chosen arbitrarily, it will follow that SAT is NP-complete. The reduction makes use of the concept of a *tableau*. A tableau for M on input x is a table that represents an *accepting* computation of M on x. (See Figure 6.) The rows of the table correspond to successive configurations in a computation of M on input x. Since M is of time complexity $p(n)$, these configurations can have length at most $m + 2$, where $m = p(n)$. Each row is formed by padding a configuration to length $p(n) + 4$ by adding trailing blanks, and then enclosing the whole in a pair of end-marker symbols (# $\cdots$ #) which are not part of the tape alphabet of M. Row 0 of the table contains the initial configuration of M, row 1 the configuration of M after one move, row 2 the configuration of M after two moves, and so on. The final row of the table (row m) is an accepting configuration of M reached after m steps. (We use the convention that once an accepting configuration is reached it is repeated, so that any accepting computation can be padded out to length exactly m.) Note that $x \in L$ if and only if a tableau for M on x exists.

The formula $\phi = \phi_M(x)$ expresses the existence of a tableau as follows: for each square (i, j) of the tableau, with $0 \le i \le m$, $0 \le j \le m+3$, introduce a set of Boolean variables Z_{ijs}, where s ranges over the set $\Gamma \cup Q \cup \{\#\}$ of tape symbols and states of M, together with the special end-marker #. The intended interpretation is that Z_{ijs} will be true if and only if square (i, j) contains the symbol s. The formula ϕ is constructed as the conjunction of four sub-formulas:

$$\phi \;=\; \phi_{\text{config}} \wedge \phi_{\text{initial}} \wedge \phi_{\text{accept}} \wedge \phi_{\text{compute}}.$$

These are defined as follows:

(i) ϕ_{config} ensures that each square contains precisely one symbol, i.e.,

$$\phi_{\text{config}} \;=\; \bigwedge_{i,j}\Big[\Big(\bigvee_{s} Z_{ijs}\Big) \;\wedge\; \bigwedge_{s \neq s'}\Big(\neg Z_{ijs} \vee \neg Z_{ijs'}\Big)\Big].$$

(ii) ϕ_{initial} ensures that the first row is the initial configuration of M on x, i.e.,

$$\phi_{\text{initial}} \;=\; Z_{00\#} \;\wedge\; Z_{01q_I} \;\wedge\; Z_{0,m+3,\#} \;\wedge\; \bigwedge_{j=2}^{n+1} Z_{0jx_{j-2}} \;\wedge\; \bigwedge_{j=n+2}^{m+2} Z_{0j\not b}.$$

(iii) ϕ_{accept} ensures that the final configuration is accepting, i.e.,

$$\phi_{\text{accept}} \;=\; \bigvee_{j} Z_{mjq_F}.$$

(iv) ϕ_{compute} ensures that every 2×3 window onto the tableau of the form

is correct, i.e.,

$$\phi_{\text{compute}} \;=\; \bigwedge_{\substack{0 \leq i \leq m-1 \\ 0 \leq j \leq m+1}} C_{ij},$$

where C_{ij} expresses correctness of the window with top left square (i, j). The clause C_{ij} is based on a fixed formula that enumerates all legitimate windows allowed by the transition relation δ of M; the various C_{ij} differ only in the values of the coordinate parameters i, j. The important point to observe is that *global* correctness of the tableau with respect to the transition relation δ follows from *local* correctness of all the 2×3 windows. [The reader should check this assertion. By placing the window so that the centre square of the top row is over a symbol of Q, one can verify that the changes that take place in the vicinity of the tape head are consistent with the transition relation; then by placing the window in positions where no symbol of Q appears in the top row, one can verify that no changes occur away from the tape head.]

Note that ϕ is essentially already in CNF. The only additional work needed is to convert the parameterized formula C_{ij} used in ϕ_{compute} into CNF. Since this formula has fixed length, a brute force method can be used.

Now ϕ has length $\mathrm{O}(p(n)^2)$ and can easily be computed in polynomial time given x. Furthermore, it follows from our discussion that ϕ is satisfiable if and only if M accepts x. $\square$

Now that we have *one* example of an NP-complete language, life becomes easier: we can demonstrate that new languages are NP-complete by exhibiting reductions from languages that are already known to be NP-complete. More precisely:

THEOREM 12.2 *Let L_1 and L_2 be languages. If L_1 is NP-hard and $L_1 \leq_{\mathrm{P}} L_2$, then L_2 is NP-hard.*

PROOF. Let L be any language in NP. Since L_1 is NP-hard, $L \leq_{\mathrm{P}} L_1$. Now the relation $\leq_{\mathrm{P}}$ is transitive [exercise], so $L \leq_{\mathrm{P}} L_2$. Thus every language L in NP is polynomial-time reducible to L_2, and hence L_2 is NP-hard. $\square$

To illustrate the use of Theorem 12.2, we shall apply it to the language CLIQUE introduced in NOTE 11.

THEOREM 12.3 CLIQUE *is NP-complete.*

PROOF. SAT is NP-complete (Cook's theorem), and SAT $\leq_{\mathrm{P}}$ CLIQUE (see NOTE 10). Hence, by Theorem 12.2, CLIQUE is NP-hard. But CLIQUE is in NP [exercise] and the theorem follows. $\square$

Using a similar approach, many other naturally occurring problems can be shown to be NP-complete; we shall meet some of these problems in NOTE 13.

We have remarked that demonstrating that a language L is NP-complete provides substantial empirical evidence that L is computationally intractable. Now, if it could be *proved* that $\mathrm{P} \subset \mathrm{NP}$, we would be *certain* that L is computationally intractable. Unfortunately, as mentioned above, no one has yet provided such a proof.