Se dice que una cláusula es de Horn si posee, a lo sumo, un literal positivo.

El problema **SAT–HORN** es el siguiente:

> *Dada una fórmula proposicional en forma normal conjuntiva tal que todas sus cláusulas son de Horn, determinar si es satisfactible.*

Probar que el problema **SAT–HORN** pertenece a la clase **P**.

Indicaciones:
(a) Téngase presente que todas las cláusulas que tengan exactamente un literal positivo son, realmente, fórmulas del tipo *implicación*.
(b) Diseñar un algoritmo determinista de coste en tiempo polinomial que, dada una fórmula de entrada φ del citado problema, devuelva una valoración σ_0 que haga verdaderas **todas** las cláusulas que son formulas del tipo implicación.
(c) En el diseño del algoritmo anterior se aconseja identificar una valoración con el conjunto de las variables a las que le asigna el valor verdadero.
(d) Probar que toda valoración τ que hace verdadera la fórmula φ, *contiene* a la valoración σ_0 (es decir, es una extensión de σ_0).
(e) Probar que la fórmula φ es satisfactible si y sólo si $\sigma_0(\varphi) = 1$.

---

normal form for two reasons: First, we know that all expressions can in principle be so represented. And second, this special form of satisfiability seems to capture the intricacy of the whole problem (as Example 4.2 perhaps indicates).

It is of interest to notice immediately that SAT can be solved in $\mathcal{O}(n^2 2^n)$ time by an exhaustive algorithm that tries all possible combinations of truth values for the variables that appear in the expression, and reports "yes" if one of them satisfies it, and "no" otherwise. Besides, SAT can be very easily solved by a *nondeterministic* polynomial algorithm, one that guesses the satisfying truth assignment and checks that it indeed satisfies all clauses; hence SAT is in **NP**. As with another important member of **NP**, TSP (D), presently we do not know whether SAT is in **P** (and we strongly suspect that it is not).

Horn Clauses

Still, there is an interesting *special case* of SAT that can be solved quite easily. We say that a clause is a *Horn clause* if it has *at most one positive literal.* That is, all its literals, except possibly for one, are negations of variables. The following clauses are therefore Horn: $(\neg x_2 \vee x_3)$, $(\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4)$, and (x_1). Of these clauses, the second is a purely negative clause (it has no positive literals), while the rest do have a positive literal, and are called *implications.* They are called so because they can be rewritten as $((x_1 \wedge x_2 \wedge \ldots \wedge x_m) \Rightarrow y)$ —where y is the positive literal. For example, the two implications among the three clauses above can be recast as follows: $(x_2 \Rightarrow x_3)$, and $(\mathbf{true} \Rightarrow x_1)$ (in the last clause, the conjunction of no variables was taken to be the "expression" **true**).

Are all these clauses satisfiable? There is an efficient algorithm for testing whether they are, based on the implicational form of Horn clauses. To make the description of the algorithm clear, it is better to consider a truth assignment not as a function from the variables to $\{\mathbf{true}, \mathbf{false}\}$, but rather as a set T of those variables that are **true**.

We wish to determine whether an expression ϕ, the conjunction of Horn clauses, is satisfiable. Initially, we only consider the implications of ϕ. The algorithm builds a satisfying truth assignment of this part of ϕ. Initially, $T := \emptyset$; that is, all variables are **false**. We then repeat the following step, until all implications are satisfied: Pick any unsatisfied implication $((x_1 \wedge x_2 \wedge \ldots \wedge x_m) \Rightarrow y)$ (that is, a clause in which all the x_is are **true** and y is **false**), and add y to T (make it **true**).

This algorithm will terminate, since T gets bigger at each step. Also, the truth assignment obtained must satisfy all implications in ϕ, since this is the only way for the algorithm to end. Finally, suppose that another truth assignment T' also satisfies all implications in ϕ; we shall show that $T \subseteq T'$. Because, if not, consider the first time during the execution of the algorithm at which T ceased being a subset of T': The clause that caused this insertion to

T cannot be satisfied by T'.

We can now determine the satisfiability of the whole expression ϕ. We claim that *ϕ is satisfiable if and only if the truth assignment T obtained by the algorithm just explained satisfies ϕ*. For suppose that there is a purely negative clause of ϕ that is not satisfied by T—say $(\neg x_1 \vee \neg x_2 \vee \ldots \vee \neg x_m)$. Thus $\{x_1, \ldots, x_m\} \subseteq T$. It follows that no superset of T can satisfy this clause, and we know that all truth assignments that satisfy ϕ are supersets of T.

Since the procedure outlined can obviously be carried out in polynomial time, we have proved the following result (where by HORNSAT we denote the satisfiability problem in the special case of Horn clauses; this is one of many special cases and variants of SAT that we shall encounter in this book).

Theorem 4.2: HORNSAT is in **P**. □

4.3 BOOLEAN FUNCTIONS AND CIRCUITS

Definition 4.3: An *n-ary Boolean function* is a function $f\{\mathbf{true}, \mathbf{false}\}^n \mapsto \{\mathbf{true}, \mathbf{false}\}$. For example, $\vee$, $\wedge$, $\Rightarrow$, and $\Leftrightarrow$ can be thought of as four of the sixteen possible binary Boolean functions, since they map pairs of truth values (those of the constituent Boolean expressions) to $\{\mathbf{true}, \mathbf{false}\}$. $\neg$ is a unary Boolean function (the only other ones are the constant functions and the identity function). More generally, any Boolean expression ϕ can be thought of as an n-ary Boolean function f_ϕ, where $n = |X(\phi)|$, since, for any truth assignment T of the variables involved in ϕ, a truth value of ϕ is defined: **true** if $T \models \phi$, and **false** if $T \not\models \phi$. Formally, we say that Boolean expression ϕ with variables $x_1, \ldots, x_n$ *expresses* the n-ary Boolean function f if, for any n-tuple of truth values $\mathbf{t} = (t_1, \ldots, t_n)$, $f(\mathbf{t})$ is **true** if $T \models \phi$, and $f(\mathbf{t})$ is **false** if $T \not\models \phi$, where $T(x_i) = t_i$ for $i = 1, \ldots, n$. □

So, every Boolean expression expresses some Boolean function. The converse is perhaps a little more interesting.

Proposition 4.3: Any n-ary Boolean function f can be expressed as a Boolean expression ϕ_f involving variables $x_1, \ldots, x_n$.

Proof: Let F be the subset of $\{\mathbf{true}, \mathbf{false}\}^n$ consisting of all n-tuples of truth values such that make f **true**. For each $\mathbf{t} = (t_1, \ldots, t_n) \in F$, let $D_\mathbf{t}$ be the conjunction of all variables x_i with $t_i =$ **true**, with all negations of variables $\neg x_i$ such that $t_i =$ **false**. Finally, the required expression is $\phi_f = \bigvee_{\mathbf{t} \in F} D_\mathbf{t}$ (notice that it is already in disjunctive normal form). It is easy to see that, for any truth assignment T appropriate to ϕ, $T \models \phi_f$ if and only if $f(\mathbf{t}) =$ **true**, where $t_i = T(x_i)$. □

The expression produced in the proof of Proposition 4.3 has length (number of symbols needed to represent it) $\mathcal{O}(n^2 2^n)$. Although many interesting Boolean functions can be represented by quite short expressions, it can be shown that *in*