Given a circuit C, it is straightforward to produce such a formula ϕ in polynomial time.

Why is the circuit C satisfiable exactly when the formula ϕ is satisfiable? If C has a satisfying assignment, then each wire of the circuit has a well-defined value, and the output of the circuit is 1. Therefore, when wire values are assigned to variables in ϕ, each clause of ϕ evaluates to 1, and thus the conjunction of all evaluates to 1. Conversely, if some assignment causes ϕ to evaluate to 1, the circuit C is satisfiable by an analogous argument. Thus, we have shown that CIRCUIT-SAT $\leq_P$ SAT, which completes the proof. ■

3-CNF satisfiability

Reducing from formula satisfiability gives us an avenue to prove many problems NP-complete. The reduction algorithm must handle any input formula, though, and this requirement can lead to a huge number of cases to consider. Instead, it is usually simpler to reduce from a restricted language of boolean formulas. Of course, the restricted language must not be polynomial-time solvable. One convenient language is 3-CNF satisfiability, or 3-CNF-SAT.

In order to define 3-CNF satisfiability, we first need to define a few terms. A ***literal*** in a boolean formula is an occurrence of a variable (such as x_1) or its negation ($\neg x_1$). A ***clause*** is the OR of one or more literals, such as $x_1 \vee \neg x_2 \vee \neg x_3$. A boolean formula is in ***conjunctive normal form***, or ***CNF***, if it is expressed as an AND of clauses, and it's in ***3-conjunctive normal form***, or ***3-CNF***, if each clause contains exactly three distinct literals.

For example, the boolean formula

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

is in 3-CNF. The first of its three clauses is $(x_1 \vee \neg x_1 \vee \neg x_2)$, which contains the three literals x_1, $\neg x_1$, and $\neg x_2$.

The language 3-CNF-SAT consists of encodings of boolean formulas in 3-CNF that are satisfiable. The following theorem shows that a polynomial-time algorithm that can determine the satisfiability of boolean formulas is unlikely to exist, even when they are expressed in this simple normal form.

Theorem 34.10
Satisfiability of boolean formulas in 3-conjunctive normal form is NP-complete.

Proof The argument from the proof of Theorem 34.9 to show that SAT $\in$ NP applies equally well here to show that 3-CNF-SAT $\in$ NP. By Lemma 34.8, therefore, we need only show that SAT $\leq_P$ 3-CNF-SAT.

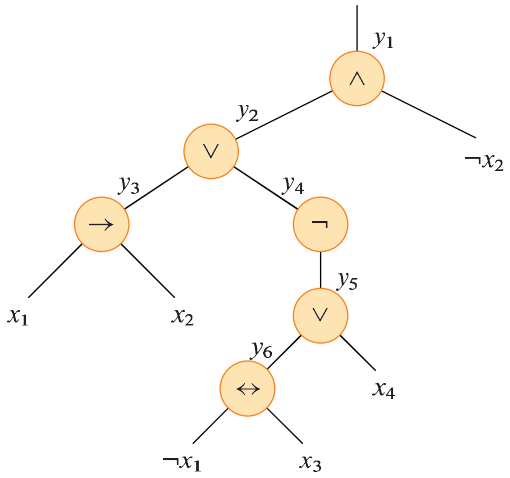

Figure 34.11 The tree corresponding to the formula $\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$.

We break the reduction algorithm into three basic steps. Each step progressively transforms the input formula ϕ closer to the desired 3-conjunctive normal form.

The first step is similar to the one used to prove CIRCUIT-SAT $\leq_P$ SAT in Theorem 34.9. First, construct a binary "parse" tree for the input formula ϕ, with literals as leaves and connectives as internal nodes. Figure 34.11 shows such a parse tree for the formula

$$\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2 \,. \tag{34.3}$$

If the input formula contains a clause such as the OR of several literals, use associativity to parenthesize the expression fully so that every internal node in the resulting tree has just one or two children. The binary parse tree is like a circuit for computing the function.

Mimicking the reduction in the proof of Theorem 34.9, introduce a variable y_i for the output of each internal node. Then rewrite the original formula ϕ as the AND of the variable at the root of the parse tree and a conjunction of clauses describing the operation of each node. For the formula (34.3), the resulting expression is

$$\begin{aligned}
\phi' \;=\; & y_1 \\
& \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\
& \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \\
& \wedge (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \\
& \wedge (y_4 \leftrightarrow \neg y_5) \\
& \wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \\
& \wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3)) \,.
\end{aligned}$$

y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	1

Figure 34.12 The truth table for the clause $(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$.

The formula ϕ' thus obtained is a conjunction of clauses ϕ'_i, each of which has at most three literals. These clauses are not yet ORs of three literals.

The second step of the reduction converts each clause ϕ'_i into conjunctive normal form. Construct a truth table for ϕ'_i by evaluating all possible assignments to its variables. Each row of the truth table consists of a possible assignment of the variables of the clause, together with the value of the clause under that assignment. Using the truth-table entries that evaluate to 0, build a formula in ***disjunctive normal form*** (or ***DNF***)—an OR of ANDs—that is equivalent to $\neg\phi'_i$. Then negate this formula and convert it into a CNF formula ϕ''_i by using ***DeMorgan's laws*** for propositional logic,

$$\neg(a \wedge b) = \neg a \vee \neg b ,$$
$$\neg(a \vee b) = \neg a \wedge \neg b ,$$

to complement all literals, change ORs into ANDs, and change ANDs into ORs.

In our example, the clause $\phi'_1 = (y_1 \leftrightarrow (y_2 \wedge \neg x_2))$ converts into CNF as follows. The truth table for ϕ'_1 appears in Figure 34.12. The DNF formula equivalent to $\neg\phi'_1$ is

$$(y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2) .$$

Negating and applying DeMorgan's laws yields the CNF formula

$$\begin{aligned}\phi''_1 = {} & (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \\ & \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2) ,\end{aligned}$$

which is equivalent to the original clause ϕ'_1.

At this point, each clause ϕ'_i of the formula ϕ' has been converted into a CNF formula ϕ''_i, and thus ϕ' is equivalent to the CNF formula ϕ'' consisting of the conjunction of the ϕ''_i. Moreover, each clause of ϕ'' has at most three literals.

The third and final step of the reduction further transforms the formula so that each clause has *exactly* three distinct literals. From the clauses of the CNF formula ϕ'', construct the final 3-CNF formula ϕ'''. This formula also uses two auxiliary variables, p and q. For each clause C_i of ϕ'', include the following clauses in ϕ''':

- If C_i contains three distinct literals, then simply include C_i as a clause of ϕ'''.
- If C_i contains exactly two distinct literals, that is, if $C_i = (l_1 \vee l_2)$, where l_1 and l_2 are literals, then include $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$ as clauses of ϕ'''. The literals p and $\neg p$ merely fulfill the syntactic requirement that each clause of ϕ''' contain exactly three distinct literals. Whether $p = 0$ or $p = 1$, one of the clauses is equivalent to $l_1 \vee l_2$, and the other evaluates to 1, which is the identity for AND.
- If C_i contains just one distinct literal l, then include $(l \vee p \vee q) \wedge (l \vee p \vee \neg q) \wedge (l \vee \neg p \vee q) \wedge (l \vee \neg p \vee \neg q)$ as clauses of ϕ'''. Regardless of the values of p and q, one of the four clauses is equivalent to l, and the other three evaluate to 1.

We can see that the 3-CNF formula ϕ''' is satisfiable if and only if ϕ is satisfiable by inspecting each of the three steps. Like the reduction from CIRCUIT-SAT to SAT, the construction of ϕ' from ϕ in the first step preserves satisfiability. The second step produces a CNF formula ϕ'' that is algebraically equivalent to ϕ'. Then the third step produces a 3-CNF formula ϕ''' that is effectively equivalent to ϕ'', since any assignment to the variables p and q produces a formula that is algebraically equivalent to ϕ''.

We must also show that the reduction can be computed in polynomial time. Constructing ϕ' from ϕ introduces at most one variable and one clause per connective in ϕ. Constructing ϕ'' from ϕ' can introduce at most eight clauses into ϕ'' for each clause from ϕ', since each clause of ϕ' contains at most three variables, and the truth table for each clause has at most $2^3 = 8$ rows. The construction of ϕ''' from ϕ'' introduces at most four clauses into ϕ''' for each clause of ϕ''. Thus the size of the resulting formula ϕ''' is polynomial in the length of the original formula. Each of the constructions can be accomplished in polynomial time. ■

Exercises

34.4-1

Consider the straightforward (nonpolynomial-time) reduction in the proof of Theorem 34.9. Describe a circuit of size n that, when converted to a formula by this method, yields a formula whose size is exponential in n.