In this section, we describe (very basic versions of) methods of solving some application problems by reduction to SAT.

1 Applications to Digital Circuits

A \( k \)-ary Boolean function is a function \( f : \{0,1\}^k \rightarrow \{0,1\} \). A combinational digital circuit is a device for computing Boolean functions, constructed by composing, or wiring together, components called “gates” that implement simple Boolean functions such as AND, OR and NOT. A circuit computing a \( k \)-ary function is said to have \( k \) inputs (or input wires) and one output. For example, the circuit of Figure 1 has three inputs, one AND gate, one OR gate, and one output. We can also construct circuits with multiple output wires: a circuit \( C \) with \( k \) input wires and \( l \) output wires computes a function \( f_C : \{0,1\}^k \rightarrow \{0,1\}^l \).

Here we illustrate simple versions of two problems involving combinational circuits. For any circuit \( C \) with \( k \) inputs and one output, we can write a formula of propositional logic that computes the same function as \( C \), using \( k \) atoms \( X_1, \ldots, X_k \) corresponding to the \( k \) input wires \( x_1, \ldots, x_k \). For each input vector \( \bar{a} \in \{0,1\}^k \), we consider \( \bar{a} \) to also be a truth assignment for the atoms with \( \bar{a}(X_i) = \text{true} \) iff \( \bar{a}(x_i) = 1 \). The formula must be satisfied by exactly the truth assignments for which the circuit outputs 1. For the circuit of Figure 1, we obtain the formula \( ((X_1 \land X_2) \lor X_3) \).

However, in our applications, we want to model the entire computation performed by a circuit with \( k \) inputs and \( l \) outputs, sometimes with reference to the internal wires. (Also note that, if we don’t have variables corresponding to internal wires, the formula might have to be much larger than the circuit.) To do this, we construct a formula with one propositional atom for each input, each output, and each internal wire that connects an output of one of the gates to the inputs of one or more other gates. We construct the
formula from sub-formulas modelling the computation of each gate within the circuit. For example, let $C$ be the circuit of Figure 1, with input wires $x_1, x_2, x_3$, output wire $y$, and internal wire $z$. The gates have semantics similar to connectives in propositional logic, with wires taking values in $\{0, 1\}$. For example, wire $z$, the output wire of the AND gate, will have the value 1 if the value applied to both of the gates input wires, $x_1$ and $x_2$, is 1, and otherwise will have the value 0. The circuit computes a Boolean function $f_C : \{0, 1\}^3 \to \{0, 1\}$. When values are applied to the input wires, the output wire will have the value $y = f_C(x_1, x_2, x_3)$.

Our formula is $\phi_C = (Z \leftrightarrow (X_1 \land X_2)) \land (Y \leftrightarrow (Z \lor X_3))$. The satisfying assignments for $\phi_C$ correspond exactly to the computations of the circuit. In particular, let $\alpha$ be truth assignment for the atoms of $\phi_C$. As before, we consider $\alpha$ also to be an assignment of values to the corresponding wires of the circuit, with $\alpha(x_1) = 1$ iff $\alpha(X_1) = true$, etc. Then the truth assignments that satisfy $\phi_C$ are exactly those for which $f_C(\alpha) = 1$ iff $\alpha(Y) = true$. That is, a truth assignment $\alpha$ satisfies $\phi_C$ iff the value $\alpha(Y)$ assigned to the “output atom” $Y$ corresponds to the value $f_C(\alpha(x_1), \alpha(x_2), \alpha(x_3))$ computed by the circuit.

1.1 Circuit Equivalence

Two circuits with the same number of inputs and outputs are equivalent if they compute the same function. Testing circuit equivalence arises often in the following form. We have a given circuit $C$, which computes a desired function $f_C$, but we hope to obtain a better circuit that computes this function, for example one using fewer gates or less energy. If we have a candidate circuit $D$, we would like to check whether $D$ is equivalent to $C$ or not. That is, we want to determine if, for every tuple $\bar{a} \in \{0, 1\}^k$, $f_D(\bar{a}) = f_C(\bar{a})$. We can do this by writing a formula $\phi_{C \neq D}$ that, intuitively, says there is an input on which the outputs of $C$ and $D$ are different.

Given two circuits $C$ and $D$, each with $n$ inputs and $m$ outputs, we can check if they compute the same function from $\{0, 1\}^n$ to $\{0, 1\}^m$ as follows. Think of a circuit that combines $C$ and $D$ as in Figure 2, producing a new circuit that computes a function from
\{0,1\}^n \rightarrow \{0,1\}^{2m}. If C and D are not equivalent, there will be an input for this combined circuit for which one of the outputs \(v_i\) of D has a different value than the corresponding output \(y_i\) of C. We write our formula \(\phi_{C \neq D}\) based on this circuit.

![Circuit for testing equivalence of \(C_A\) and \(C_B\).]

Suppose we have two formulas, \(\phi_C\) and \(\phi_D\), which model C and D (respectively), as described earlier in this section. Suppose that \(\phi_C\) has atoms \(X_1, \ldots, X_n\), for the inputs to C, atoms \(Y_1, \ldots, Y_m\), for the outputs of C, and atoms \(Z_1, \ldots, Z_k\) for the internal wires of C. Further assume (or, by renaming atoms, arrange that) formula \(\phi_D\) has the same atoms \(X_1, \ldots, X_n\) corresponding to inputs \(x_1, \ldots, x_n\), atoms \(V_1, \ldots, V_k\) corresponding to outputs \(v_1, \ldots, v_k\) (with, of course, each \(v_i\) corresponding to \(y_i\) in the output of C), and atoms \(W_1, \ldots, W_j\) corresponding to the internal wires of D.

Now, a formula that models the computation of this circuit is just \((\phi_C \land \phi_D)\). To obtain a formula \(\phi_{C \neq D}\) that says there is an input to the combined circuit for which the outputs of C and D are different, we simply conjoin to this the statement that the outputs are different, obtaining as \(\phi_{C \neq D}\) the formula

\[
\phi_C \land \phi_D \land \neg((Y_1 \leftrightarrow V_1) \land (Y_2 \leftrightarrow V_2) \land \ldots \land (Y_m \leftrightarrow V_m)).
\]

### 1.2 Automated Test Pattern Generation (ATPG)

In this application, we suppose we are manufacturing chips containing a digital circuit. The manufacturing process is imperfect, and some chips will be flawed. We want to construct tests that detect the most likely flaws in the chips. A common kind of flaw is known as a “single stuck at fault”, in which the effect of a chip flaw is that a particular wire in the circuit has a fixed value. To illustrate, suppose that in chips implementing the circuit of Figure 1, sometimes there is a flaw which has the wire \(z\) stuck at 0. The flawed version of the circuit is as shown in Figure 3.
We want to generate an input for which the correct and flawed circuits will produce different outputs. We can use this to test the chips for the flaw, because on this input the correct chips will give a different output than the flawed chips.

We proceed much as in the equivalence case, but here the circuits are partly the same, and we need only model the way in which they are different. To do this, we make a formula $\phi_{flaw}$ which models just that part of the circuit affected by the flaw. The only wires that may have values different from the corresponding wires in the correct circuit are $z'$ and $y'$. So, $\phi_{flaw}$ is $\lnot Z' \land (Y' \leftrightarrow (Z' \lor X_3))$.

Now, we obtain the following formula, $\phi_{test}$, that says there is an input on which the correct and flawed circuits have different output:

$$\phi_C \land \phi_{flaw} \land \lnot (Y \leftrightarrow Y').$$

A satisfying assignment for $\phi_{test}$ gives an input (the values given to $x_1, x_2$ and $x_3$) which constitutes a test pattern for the flaw: a flawed chip will give a different output value than a chip with no flaws.

Figure 3: A circuit with a flawed wire stuck at 0.