As another example let us check the formula **AF**green. Initially s_1 is assigned to the variable *NextStates*, but at the first iteration PRE-IMG-AF returns the empty set, then the loop terminates and the algorithm returns *False*. ■

C.5 Symbolic Model Checking

Most often, realistic models of systems need huge numbers of states. For example, an interlocking system may have something like 10^{200} states. Symbolic model checking [99] has been devised to deal with large state spaces. It is a form of model checking in which propositional formulas are used for the compact representation of finite-state models, and transformations over propositional formulas provide a basis for efficient exploration of the state space. Most often, the use of symbolic techniques allows for the analysis of large systems, even systems with 10^{200} states [99]. The fundamental ideas of symbolic model checking are the following.

1. Model checking is performed by exploring sets of states, rather than single states.
2. The model checking problem is represented symbolically: the sets of states and the transitions over sets of states are represented by logical formulas.

In order to represent a model checking problem symbolically, we need to represent symbolically the sets of states of a Kripke Structure, its transition relation, and the model checking algorithms.

Symbolic Representation of Sets of States. A vector of distinct propositional variables $\boldsymbol{x}$, called *state variables*, is devoted to the representation of the states of a Kripke structure. Each of these variables has a direct association with a proposition symbol of $\mathcal{P}$. Therefore, in the rest of this section we will not distinguish between a proposition and the corresponding propositional variable. For instance, in the Kripke structure in Example C.2 (see page 563), $\boldsymbol{x}$ is the vector (green, open). A state is the set of propositions of $\mathcal{P}$ that hold in the state. For each state s, there is a corresponding assignment of truth values to the state variables in $\boldsymbol{x}$: each variable in s is *True*, and all other variables are *False*. We represent s with a propositional formula $\xi(s)$, whose unique satisfying assignment of truth values corresponds to s. For instance, the formula representing state s_1 in Example C.2 is green $\wedge \neg$open. This representation naturally extends to any set of states $Q \subseteq S$ as follows:

$$\xi(Q) = \bigvee_{s \in Q} \xi(s)$$

That is, we associate a set of states with the disjunction of the formulas representing each of the states. The satisfying assignments of $\xi(Q)$ are exactly the assignments representing the states of Q.

A remark is in order. We are using a propositional formula to represent the set of assignments that satisfy it (and hence to represent the corresponding set of states), but we do not care about the actual syntax of the formula used, and thus in the following discussion we will not distinguish among equivalent formulas that represent the same sets of assignments. Although the actual syntax of the formula may have a computational impact, in the next section we will show that the use of formulas for representing sets of states is indeed practical.

The main efficiency of the symbolic representation is that the cardinality of the represented set is not directly related to the size of the formula. For instance, $\xi(2^{\mathcal{P}})$ and $\xi(\emptyset)$ are the formulas *True* and *False*, respectively, independent of the cardinality of $\mathcal{P}$.

As a further advantage, the symbolic representation can provide an easy way to ignore irrelevant information. For instance, notice that the variable open does not appear in the formula $\xi(\{s_0, s_2\}) = \neg$green. For this reason, a symbolic representation can have a dramatic improvement over an explicit state representation that enumerates the states of the Kripke Structure. This is what allows symbolic model checkers to handle finite-state models that have very large numbers of states (see, e.g., [99]).

Another advantage of the symbolic representation is the natural encoding of set-theoretic transformations (e.g., union, intersection, complementation) into propositional operations, as follows:

$$\xi(Q_1 \cup Q_2) = \xi(Q_1) \vee \xi(Q_2)$$
$$\xi(Q_1 \cap Q_2) = \xi(Q_1) \wedge \xi(Q_2)$$
$$\xi(S - Q) = \xi(S) \wedge \neg\xi(Q)$$

Symbolic Representation of Transition Relations. We represent transition relations through the vector of state variables $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$ and a further vector $\boldsymbol{x}' = \langle x'_1, \ldots, x'_n \rangle$ of propositional variables, called *next-state variables*. We write $\xi'(s)$ for the representation of the state s in the next-state variables. $\xi'(Q)$ is the formula corresponding to the set of states Q. In the following, $\Phi[\boldsymbol{x} \leftarrow \boldsymbol{y}]$ is the parallel substitution in formula Φ of the variables in vector $\boldsymbol{x}$ with the corresponding variables in $\boldsymbol{y}$. We define the representation of a set of states in the next variables as follows:

$$\xi'(s) = \xi(s)[\boldsymbol{x} \leftarrow \boldsymbol{x}'].$$

We call the operation $\Phi[\boldsymbol{x} \leftarrow \boldsymbol{x}']$ *forward shifting* because it transforms the representation of a set of current states in the representation of a set of next states. The dual operation $\Phi[\boldsymbol{x}' \leftarrow \boldsymbol{x}]$ is called *backward shifting*. In the following, we call the variables in $\boldsymbol{x}$ the *current-state variables* to distinguish them from the next-state variables.

For the interlocking example in Figure C.3, the single transition from state s_0 to state s_1 is represented by the formula

$$\xi(\langle s_0, s_1\rangle) = \xi(s_0) \wedge \xi'(s_1),$$

that is,

$$\xi(\langle s_0, s_1\rangle) = (\neg\mathsf{green} \wedge \neg\mathsf{open}) \wedge (\mathsf{green}' \wedge \neg\mathsf{open}')$$

The transition relation R of a Kripke Structure is a set of transitions and is thus represented by the formula in the variables of $\boldsymbol{x}$ and of $\boldsymbol{x}'$,

$$\xi(R) = \bigvee_{r \in R} \xi(r),$$

in which each satisfying assignment represents a possible transition.

Symbolic Representation of Model Checking Algorithms. In order to make explicit that the formula $\xi(Q)$ contains the variables $x_1, \ldots, x_n$ of $\boldsymbol{x}$, in the following we will use the expression $Q(\boldsymbol{x})$ to mean $\xi(Q)$. Similarly, we will use the expression $Q(\boldsymbol{x}')$ to mean $\xi'(Q)$. Let $S(\boldsymbol{x})$, $R(\boldsymbol{x}, \boldsymbol{x}')$, and $S_0(\boldsymbol{x})$ be the formulas representing the states, the transition relation, and the initial states of a Kripke structure, respectively.

In the following, we will use quantification in the style of the logic of Quantified Boolean Formulas (QBFs). QBFs are a definitional extension to propositional logic, in which propositional variables can be universally and existentially quantified. If Φ is a formula and v_i is one of its variables, then the existential quantification of v_i in Φ, written $\exists v_i.\, \Phi(v_1, \ldots, v_n)$, is equivalent to $\Phi(v_1, \ldots, v_n)[v_i \leftarrow \mathit{False}] \vee \Phi(v_1, \ldots, v_n)[v_i \leftarrow \mathit{True}]$. Analogously, the universal quantification $\forall v_i.\, \Phi(v_1, \ldots, v_n)$ is equivalent to $\Phi(v_1, \ldots, v_n)[v_i \leftarrow \mathit{False}] \wedge \Phi(v_1, \ldots, v_n)[v_i \leftarrow \mathit{True}]$. QBFs allow for an exponentially more compact representation than propositional formulas.

The symbolic representation of the *image* of a set of states Q, i.e., the set of states reachable from any state in Q with one state transition, is the result of applying the substitution $[\boldsymbol{x}' \leftarrow \boldsymbol{x}]$ to the formula $\exists \boldsymbol{x}.\, (R(\boldsymbol{x}, \boldsymbol{x}') \wedge Q(\boldsymbol{x}))$:

$$(\exists \boldsymbol{x}.\, (R(\boldsymbol{x}, \boldsymbol{x}') \wedge Q(\boldsymbol{x})))[\boldsymbol{x}' \leftarrow \boldsymbol{x}]$$

Notice that, with this single operation, we symbolically simulate the transition from any of the states in Q. The dual backward image is the following:

$$(\exists \boldsymbol{x}'.\, (R(\boldsymbol{x}, \boldsymbol{x}') \wedge Q(\boldsymbol{x}')))$$

From the definition of PRE-IMG-EF(Q) (see Equation C.2), we have therefore that ξ(PRE-IMG-EF(Q)) is:

$$\exists \boldsymbol{x}'.\,(R(\boldsymbol{x},\boldsymbol{x}') \wedge Q(\boldsymbol{x}'))$$

while ξ(PRE-IMG-AF(Q)) (see Equation C.3) is:

$$\forall \boldsymbol{x}'.\,(R(\boldsymbol{x},\boldsymbol{x}') \rightarrow Q(\boldsymbol{x}'))$$

In both cases, the resulting formula is obtained as a one-step computation and can often describe compactly a large set of states.

Given the basic building blocks just defined, the algorithms presented in the previous section can be symbolically implemented by replacing, within the same control structure, each function call with the symbolic counterpart and by casting the operations on sets into the corresponding operations on propositional formulas.

C.6 BDD-Based Symbolic Model Checking

BDDs provide a way to implement the symbolic representation mechanisms presented in the previous section (e.g., tautology checking, quantification, shifting).

A BDD is a directed acyclic graph (DAG). The terminal nodes are either *True* or *False* (alternatively indicated with 0 and 1, respectively). Each nonterminal node is associated with a Boolean variable and with two BDDs that are called the *left* and *right branches*. Figure C.5 shows some simple BDDs for the interlocking example. At each nonterminal node, the right or left branch is depicted as a solid or dashed line and represents the assignment of the value *True* or *False* to the corresponding variable.

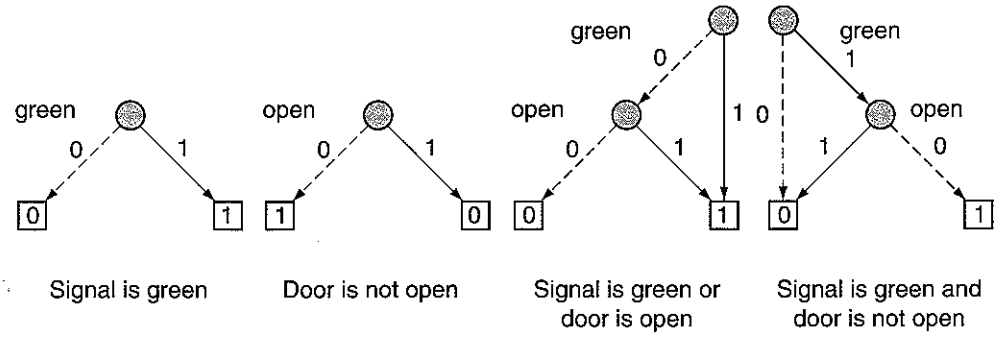

Figure C.5 BDDs for the Boolean formulas green, ¬open, green ∨ open, and green ∧ ¬open.

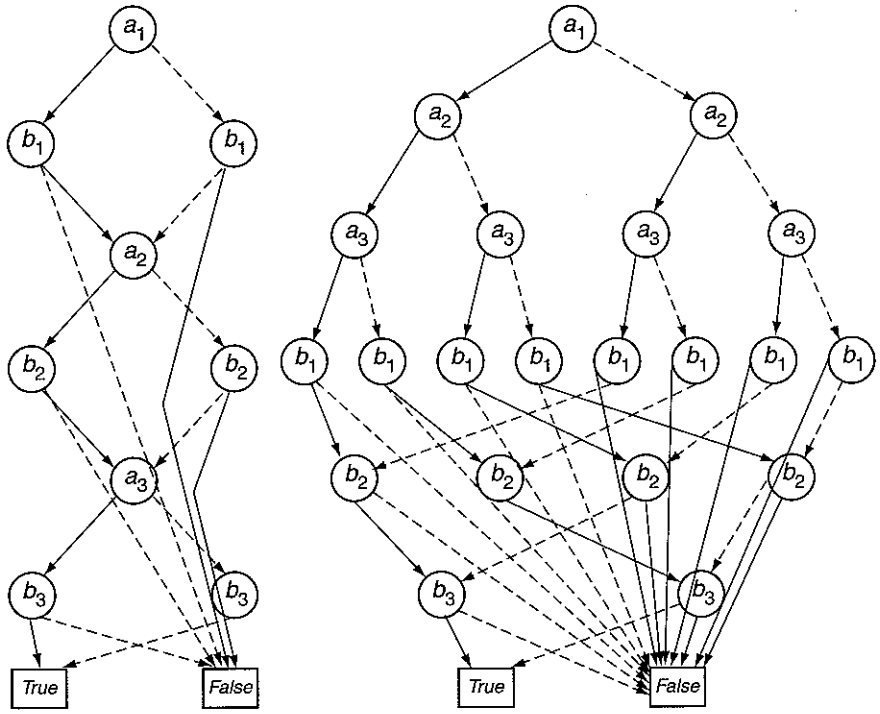

Figure C.6 Two BDDs for the formula $(a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2) \wedge (a_3 \leftrightarrow b_3)$.

Given a BDD, the value corresponding to a given truth assignment to the variables is determined by traversing the graph from the root to the leaves, following each branch indicated by the value assigned to the variables. A path from the root to a leaf can visit nodes associated with a subset of all the variables of the BDD. The reached leaf node is labeled with the resulting truth value. If v is a BDD, its size $|v|$ is the number of its nodes. If n is a node, we use $var(n)$ to denote the variable indexing node n. BDDs are a canonical representation of Boolean formulas if (1) there is a total order $<$ over the set of variables used to label nodes, such that for any node n and respective nonterminal child m, their variables must be ordered, i.e., $var(n) < var(m)$; and (2) the BDD contains no subgraphs that are isomorphic to the BDD itself.

The choice of variable ordering may have a dramatic impact on the dimension of a BDD. For example, Figure C.6 depicts two BDDs for the same formula $(a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2) \wedge (a_3 \leftrightarrow b_3)$ obtained with different variable orderings.

BDDs can be used to compute the results of applying the usual Boolean operators. Given a BDD that represents a formula, it is possible to transform it to obtain the BDD representing the negation of the formula. Given two BDDs representing

two formulas, it is possible to combine them to obtain the BDD representing the conjunction or the disjunction of the two formulas. For instance, Figure C.5 shows how the BDD representing the formula green $\wedge$ $\neg$open can be obtained from the BDDs representing the formulas green and $\neg$open. Substitution and quantification on Boolean formulas can also be performed as BDD transformations.