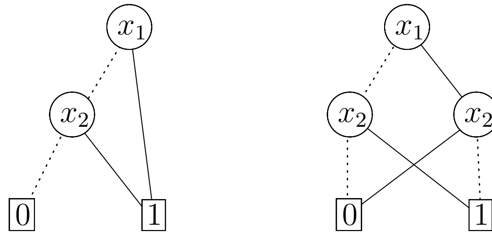


**Figure 6.26.** Two OBDDs for the set $\{s_0, s_1\}$ (Example 6.12).

of $S$ or not; so we may choose to include it or not in order to optimise the size of the OBDD. For example, the subset $\{s_0, s_1\}$ is better represented by the boolean function $x_1 + x_2$, since its OBDD is smaller than that for $x_1 \cdot \overline{x_2} + \overline{x_1} \cdot x_2$ (Figure 6.26).

In order to justify the claim that the representation of subsets of $S$ as OBDDs will be suitable for the algorithm presented in Section 3.6.1, we need to look at how the operations on subsets which are used in that algorithm can be implemented in terms of the operations we have defined on OBDDs. The operations in that algorithm are:

- Intersection, union and complementation of subsets. It is clear that these are represented by the boolean functions $\cdot$, $+$ and $\bar{}$ respectively. The implementation via OBDDs of $\cdot$ and $+$ uses the `apply` algorithm (Section 6.2.2).
- The functions

$$\begin{aligned} \text{pre}_\exists(X) &= \{s \in S \mid \text{exists } s', (s \to s' \text{ and } s' \in X)\} \\ \text{pre}_\forall(X) &= \{s \mid \text{for all } s', (s \to s' \text{ implies } s' \in X)\}. \end{aligned} \tag{6.4}$$

  The function $\text{pre}_\exists$ (instrumental in $\texttt{SAT}_{\texttt{EX}}$ and $\texttt{SAT}_{\texttt{EU}}$) takes a subset $X$ of states and returns the set of states which can make a transition into $X$. The function $\text{pre}_\forall$, used in $\texttt{SAT}_{\texttt{AF}}$, takes a set $X$ and returns the set of states which can make a transition *only* into $X$. In order to see how these are implemented in terms of OBDDs, we need first to look at how the transition relation itself is represented.

### 6.3.2 Representing the transition relation

The transition relation $\to$ of a model $\mathcal{M} = (S, \to, L)$ is a subset of $S \times S$. We have already seen that subsets of a given finite set may be represented as OBDDs by considering the characteristic function of a binary encoding.

Just like in the case of subsets of $S$, the binary encoding is naturally given by the labelling function $L$. Since $\to$ is a subset of $S \times S$, we need two copies of the boolean vectors. Thus, the link $s \to s'$ is represented by the pair of

| $x_1$ | $x_2$ | $x'_1$ | $x'_2$ | $\rightarrow$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

| $x_1$ | $x'_1$ | $x_2$ | $x'_2$ | $\rightarrow$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

**Figure 6.27.** The truth table for the transition relation of Figure 6.24 (see Example 6.13). The left version shows the ordering of variables $[x_1, x_2, x'_1, x'_2]$, while the right one orders the variables $[x_1, x'_1, x_2, x'_2]$ (the rows are ordered lexicographically).

boolean vectors $((v_1, v_2, \ldots, v_n), (v'_1, v'_2, \ldots, v'_n))$, where $v_i$ is 1 if $p_i \in L(s)$ and 0 otherwise; and similarly, $v'_i$ is 1 if $p_i \in L(s')$ and 0 otherwise. As an OBDD, the link is represented by the OBDD for the boolean function

$$(l_1 \cdot l_2 \cdot \cdots \cdot l_n) \cdot (l'_1 \cdot l'_2 \cdot \cdots \cdot l'_n)$$

and a set of links (for example, the entire relation $\rightarrow$) is the OBDD for the $+$ of such formulas.

**Example 6.13** To compute the OBDD for the transition relation of Figure 6.24, we first show it as a truth table (Figure 6.27 (left)). Each 1 in the final column corresponds to a link in the transition relation and each 0 corresponds to the absence of a link. The boolean function is obtained by taking the disjunction of the rows having 1 in the last column and is

$$f^{\rightarrow} \stackrel{\text{def}}{=} \overline{x}_1 \cdot \overline{x}_2 \cdot \overline{x}'_1 \cdot \overline{x}'_2 + \overline{x}_1 \cdot \overline{x}_2 \cdot x'_1 \cdot \overline{x}'_2 + x_1 \cdot \overline{x}_2 \cdot \overline{x}'_1 \cdot x'_2 + \overline{x}_1 \cdot x_2 \cdot \overline{x}'_1 \cdot \overline{x}'_2. \tag{6.5}$$

It turns out that it is usually more efficient to interleave unprimed and primed variables in the OBDD variable ordering for $\rightarrow$. We therefore use

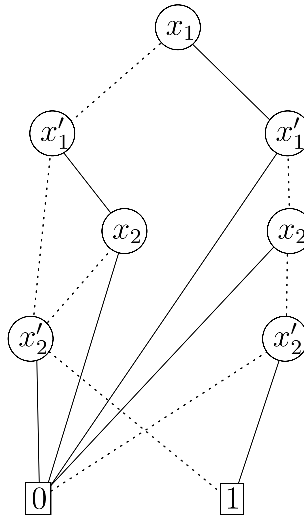


**Figure 6.28.** An OBDD for the transition relation of Example 6.13.

$[x_1, x_1', x_2, x_2']$ rather than $[x_1, x_2, x_1', x_2']$. Figure 6.27 (right) shows the truth table redrawn with the interleaved ordering of the columns and the rows reordered lexicographically. The resulting OBDD is shown in Figure 6.28.

### 6.3.3 Implementing the functions pre$_\exists$ and pre$_\forall$

It remains to show how an OBDD for $\text{pre}_\exists(X)$ and $\text{pre}_\forall(X)$ can be computed, given OBDDs $B_X$ for $X$ and $B_\rightarrow$ for the transition relation $\rightarrow$. First we observe that pre$_\forall$ can be expressed in terms of complementation and pre$_\exists$, as follows: $\text{pre}_\forall(X) = S - \text{pre}_\exists(S - X)$, where we write $S - Y$ for the set of all $s \in S$ which are not in $Y$. Therefore, we need only explain how to compute the OBDD for $\text{pre}_\exists(X)$ in terms of $B_X$ and $B_\rightarrow$. Now (6.4) suggests that one should proceed as follows:

1. Rename the variables in $B_X$ to their primed versions; call the resulting OBDD $B_{X'}$.
2. Compute the OBDD for $\texttt{exists}(\hat{x}', \texttt{apply}(\cdot, B_\rightarrow, B_{X'}))$ using the `apply` and `exists` algorithms (Sections 6.2.2 and 6.2.4).

### 6.3.4 Synthesising OBDDs

The method used in Example 6.13 for producing an OBDD for the transition relation was to compute first the truth table and then an OBDD which might not be in its fully reduced form; hence the need for a final call to

the `reduce` function. However, this procedure would be unacceptable if applied to realistically sized systems with a large number of variables, for the truth table's size is exponential in the number of boolean variables. The key idea and attraction of applying OBDDs to finite systems is therefore to take a system description in a language such as SMV and to synthesise the OBDD directly, without having to go via intermediate representations (such as binary decision *trees* or truth tables) which are exponential in size.

SMV allows us to define the next value of a variable in terms of the current values of variables (see the examples of code in Section 3.3.2)[3]. This can be compiled into a set of boolean functions $f_i$, one for each variable $x_i$, which define the next value of $x_i$ in terms of the current values of all the variables. In order to cope with non-deterministic assignment (such as the assignment to `status` in the example on page 192), we extend the set of variables by adding unconstrained variables which model the input. Each $x'_i$ is a deterministic function of this enlarged set of variables; thus, $x'_i \leftrightarrow f_i$, where $f \leftrightarrow g = 1$ if, and only if, $f$ and $g$ compute the same values, i.e. it is a shorthand for $\overline{f} \oplus g$.

The boolean function representing the transition relation is therefore of the form

$$\prod_{1 \leq i \leq n} x'_i \leftrightarrow f_i, \tag{6.6}$$

where $\prod_{1 \leq i \leq n} g_i$ is a shorthand for $g_1 \cdot g_2 \cdot \ldots \cdot g_n$. Note that the $\prod$ ranges only over the non-input variables. So, if $u$ is an input variable, the boolean function does not contain any $u' \leftrightarrow f_u$.

Figure 6.22 showed how the reduced OBDD could be computed from the parse tree of such a boolean function. Thus, it is possible to compile SMV programs into OBDDs such that their specifications can be executed according to the pseudo-code of the function `SAT`, now interpreted over OBDDs. On page 396 we will see that this OBDD implementation can be extended to simple fairness constraints.

**Modelling sequential circuits** As a further application of OBDDs to verification, we show how OBDDs representing circuits may be synthesised.

*Synchronous circuits.* Suppose that we have a design of a sequential circuit such as the one in Figure 6.29. This is a synchronous circuit (meaning that

[3] SMV also allows next values to be defined in terms of next values, i.e. the keyword `next` to appear in expressions on the right-hand side of :=. This is useful for describing synchronisations, for example, but we ignore that feature here.

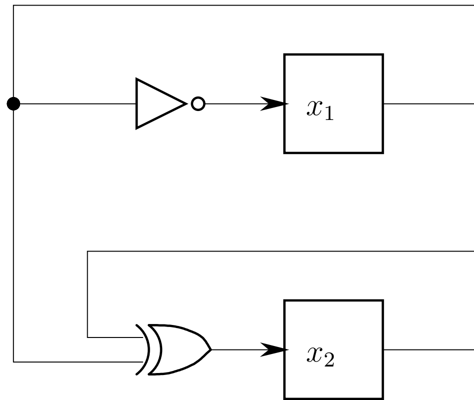


**Figure 6.29.** A simple synchronous circuit with two registers.

all the state variables are updated synchronously in parallel) whose functionality can be described by saying what the values of the registers $x_1$ and $x_2$ in the next state of the circuit are. The function $f^{\rightarrow}$ coding the possible next states of the circuits is

$$(x_1' \leftrightarrow \overline{x}_1) \cdot (x_2' \leftrightarrow x_1 \oplus x_2). \tag{6.7}$$

This may now be translated into an OBDD by the methods summarised in Figure 6.22.

*Asynchronous circuits.* The symbolic encoding of synchronous circuits is in its logical structure very similar to the encoding of $f^{\rightarrow}$ for CTL models; compare the codings in (6.7) and (6.6). In asynchronous circuits, or processes in SMV, the logical structure of $f^{\rightarrow}$ changes. As before, we can construct functions $f_i$ which code the possible next state in the *local component*, or the SMV process, $i$. For asynchronous systems, there are two principal ways of composing these functions into global system behaviour:

- In a *simultaneous model*, a global transition is one in which any number of components may make their local transition. This is modelled as

$$f^{\rightarrow} \stackrel{\text{def}}{=} \prod_{i=1}^{n} \left((x_i' \leftrightarrow f_i) + (x_i' \leftrightarrow x_i)\right). \tag{6.8}$$

- In an *interleaving model*, exactly one local component makes a local transition;

all other local components remain in their local state:

$$f^{\rightarrow} \stackrel{\text{def}}{=} \sum_{i=1}^{n} \left( (x_i' \leftrightarrow f_i) \cdot \prod_{j \neq i} (x_j' \leftrightarrow x_j) \right). \tag{6.9}$$

Observe the duality in these approaches: the simultaneous model has an outer product, whereas the interleaving model has an outer sum. The latter, if used in $\exists \hat{x}'.f$ ('for some next state'), can be optimised since sums distribute over existential quantification; in Chapter 2 this was the equivalence $\exists x.(\phi \vee \psi) \equiv \exists x.\phi \vee \exists x.\psi$. Thus, global states reachable in one step are the 'union' of all the states reachable in one step in the local components; compare the formulas in (6.8) and (6.9) with (6.6).

## 6.4 A relational mu-calculus

We saw in Section 3.7 that evaluating the set of states satisfying a CTL formula in a model may involve the computation of a fixed point of an operator. For example, $[\![\mathrm{EF}\,\phi]\!]$ is the least fixed point of the operator $F \colon \mathcal{P}(S) \to \mathcal{P}(S)$ given by $F(X) = [\![\phi]\!] \cup \mathrm{pre}_{\exists}(X)$.

In this section, we introduce a syntax for referring to fixed points in the context of boolean formulas. Fixed-point invariants frequently occur in all sorts of applications (for example, the common-knowledge operator $C_G$ in Chapter 5), so it makes sense to have an intermediate language for expressing such invariants syntactically. This language also provides a formalism for describing interactions and dependences of such invariants. We will see shortly that symbolic model checking in the presence of simple fairness constraints exhibits such more complex relationships between invariants.

### 6.4.1 Syntax and semantics

**Definition 6.14** The formulas of the relational mu-calculus are given by the grammar

$$\begin{aligned} v &::= x \mid Z \\ f &::= 0 \mid 1 \mid v \mid \overline{f} \mid f_1 + f_2 \mid f_1 \cdot f_2 \mid f_1 \oplus f_2 \mid \\ &\qquad \exists x.f \mid \forall x.f \mid \mu Z.f \mid \nu Z.f \mid f[\hat{x} := \hat{x}'] \end{aligned} \tag{6.10}$$

where $x$ and $Z$ are boolean variables, and $\hat{x}$ is a tuple of variables. In the formulas $\mu Z.f$ and $\nu Z.f$, any occurrence of $Z$ in $f$ is required to fall within an even number of complementation symbols $^-$; such an $f$ is said to be formally monotone in $Z$. (In exercise 7 on page 410 we consider what happens if we do not require formal monotonicity.)

**Convention 6.15** The binding priorities for the grammar in (6.10) are that $\bar{}$, and $[\hat{x} := \hat{x}']$ have the highest priority; followed by $\exists x$ and $\forall y$; then $\mu Z$ and $\nu Z$; followed by $\cdot$ . The operators $+$ and $\oplus$ have the lowest binding priority.

The symbols $\mu$ and $\nu$ are called *least fixed-point* and *greatest fixed-point* operators, respectively. In the formula $\mu Z.f$, the interesting case is that in which $f$ contains an occurrence of $Z$. In that case, $f$ can be thought of as a function, taking $Z$ to $f$. The formula $\mu Z.f$ is intended to mean the least fixed point of that function. Similarly, $\nu Z.f$ is the greatest fixed point of the function. We will see how this is done in the semantics.

The formula $f[\hat{x} := \hat{x}']$ expresses an explicit substitution which forces $f$ to be evaluated using the values of $x_i'$ rather than $x_i$. (Recall that the primed variables refer to the next state.) Thus, this syntactic form is not a meta-operation denoting a substitution, but an explicit syntactic form in its own right. The substitution will be made on the semantic side, not the syntactic side. This difference will become clear when we present the semantics of $\models$.

A valuation $\rho$ for $f$ is an assignment of values 0 or 1 to all variables $v$. We define a *satisfaction relation* $\rho \models f$ inductively over the structure of such formulas $f$, given a valuation $\rho$.

**Definition 6.16** Let $\rho$ be a valuation and $v$ a variable. We write $\rho(v)$ for the value of $v$ assigned by $\rho$. We define $\rho[v \mapsto 0]$ to be the updated valuation which assigns 0 to $v$ and $\rho(w)$ to all other variables $w$. Dually, $\rho[v \mapsto 1]$ assigns 1 to $v$ and $\rho(w)$ to all other variables $w$.

For example, if $\rho$ is the valuation represented by $(x, y, Z) \Rightarrow (1, 0, 1)$ – meaning that $\rho(x) = 1$, $\rho(y) = 0$, $\rho(Z) = 1$ and $\rho(v) = 0$ for all other variables $v$ – then $\rho[x \mapsto 0]$ is represented by $(x, y, Z) \Rightarrow (0, 0, 1)$, whereas $\rho[Z \mapsto 0]$ is $(x, y, Z) \Rightarrow (1, 0, 0)$. The assumption that valuations assign values to all variables is rather mathematical, but avoids some complications which have to be addressed in implementations (see exercise 3 on page 409).

Updated valuations allow us to define the satisfaction relation for all formulas without fixed points:

**Definition 6.17** We define a satisfaction relation $\rho \models f$ for formulas $f$ without fixed-point subformulas with respect to a valuation $\rho$ by structural induction:

- $\rho \not\models 0$
- $\rho \models 1$
- $\rho \models v$ iff $\rho(v)$ equals 1

- $\rho \vDash \overline{f}$ iff $\rho \nvDash f$
- $\rho \vDash f + g$ iff $\rho \vDash f$ or $\rho \vDash g$
- $\rho \vDash f \cdot g$ iff $\rho \vDash f$ and $\rho \vDash g$
- $\rho \vDash f \oplus g$ iff $\rho \vDash (f \cdot \overline{g} + \overline{f} \cdot g)$
- $\rho \vDash \exists x.f$ iff $\rho[x \mapsto 0] \vDash f$ or $\rho[x \mapsto 1] \vDash f$
- $\rho \vDash \forall x.f$ iff $\rho[x \mapsto 0] \vDash f$ and $\rho[x \mapsto 1] \vDash f$
- $\rho \vDash f[\hat{x} := \hat{x}']$ iff $\rho[\hat{x} := \hat{x}'] \vDash f$,

where $\rho[\hat{x} := \hat{x}']$ is the valuation which assigns the same values as $\rho$, but for each $x_i$ it assigns $\rho(x_i')$.

The semantics of boolean quantification closely resembles the one for the quantifiers of predicate logic. The crucial difference, however, is that boolean formulas are only interpreted over the fixed universe of values $\{0, 1\}$, whereas predicate formulas may take on values in all sorts of finite or infinite models.

**Example 6.18** Let $\rho$ be such that $\rho(x_1')$ equals 0 and $\rho(x_2')$ is 1. We evaluate $\rho \vDash (x_1 + \overline{x}_2)[\hat{x} := \hat{x}']$ which holds iff $\rho[\hat{x} := \hat{x}'] \vDash (x_1 + \overline{x}_2)$. Thus, we need $\rho[\hat{x} := \hat{x}'] \vDash x_1$ or $\rho[\hat{x} := \hat{x}'] \vDash \overline{x}_2$ to be the case. Now, $\rho[\hat{x} := \hat{x}'] \vDash x_1$ cannot be, for this would mean that $\rho(x_1')$ equals 1. Since $\rho[\hat{x} := \hat{x}'] \vDash \overline{x}_2$ would imply that $\rho[\hat{x} := \hat{x}'] \nvDash x_2$, we infer that $\rho[\hat{x} := \hat{x}'] \nvDash \overline{x}_2$ because $\rho(x_2')$ equals 1. In summary, we demonstrated that $\rho \nvDash (x_1 + \overline{x}_2)[\hat{x} := \hat{x}']$.

We now extend the definition of $\vDash$ to the fixed-point operators $\mu$ and $\nu$. Their semantics will have to reflect their meaning as least, respectively greatest, fixed-point operators. We define the semantics of $\mu Z.f$ via its syntactic approximants which unfold the meaning of $\mu Z.f$:

$$\begin{aligned} \mu_0 Z.f &\stackrel{\text{def}}{=} 0 \\ \mu_{m+1} Z.f &\stackrel{\text{def}}{=} f[\mu_m Z.f/Z] \qquad (m \geq 0). \end{aligned} \tag{6.11}$$

The unfolding is achieved by a meta-operation $[g/Z]$ which, when applied to a formula $f$, replaces all free occurrences of $Z$ in $f$ with $g$. Thus, we view $\mu Z$ as a binding construct similar to the quantifiers $\forall x$ and $\exists x$, and $[g/Z]$ is similar to the substitution $[t/x]$ in predicate logic. For example, $(x_1 + \exists x_2.(Z \cdot x_2))[\overline{x}_1/Z]$ is the formula $x_1 + \exists x_2.(\overline{x}_1 \cdot x_2)$, whereas $((\mu Z.x_1 + Z) \cdot (x_1 + \exists x_2.(Z \cdot x_2)))[\overline{x}_1/Z]$ equals $(\mu Z.x_1 + Z) \cdot (x_1 + \exists x_2.(\overline{x}_1 \cdot x_2))$. See exercise 3 on page 409 for a formal account of this meta-operation.

With these approximants we can define:

$$\rho \vDash \mu Z.f \text{ iff } (\rho \vDash \mu_m Z.f \text{ for some } m \geq 0). \tag{6.12}$$

Thus, to determine whether $\mu Z.f$ is true with respect to a valuation $\rho$, we have to find some $m \geq 0$ such that $\rho \vDash \mu_m Z.f$ holds. A sensible strategy is to try to prove this for the smallest such $m$ possible, if indeed such an $m$ can be found. For example, in attempting to show $\rho \vDash \mu Z.Z$, we try $\rho \vDash \mu_0 Z.Z$, which fails since the latter formula is just 0. Now, $\mu_1 Z.Z$ is defined to be $Z[\mu_0 Z.Z/Z]$ which is just $\mu_0 Z.Z$ again. We can now use mathematical induction on $m \geq 0$ to show that $\mu_m Z.Z$ equals $\mu_0 Z.Z$ for all $m \geq 0$. By (6.12), this implies $\rho \nvDash \mu Z.Z$.

The semantics for $\nu Z.f$ is similar. First, let us define a family of approximants $\nu_0 Z.f$, $\nu_1 Z.f$, ... by

$$\begin{aligned} \nu_0 Z.f &\stackrel{\text{def}}{=} 1 \\ \nu_{m+1} Z.f &\stackrel{\text{def}}{=} f[\nu_m Z.f/Z] \qquad (m \geq 0). \end{aligned} \tag{6.13}$$

Note that this definition only differs from the one for $\mu_m Z.f$ in that the first approximant is defined to be 1 instead of 0.

Recall how the greatest fixed point for EG $\phi$ requires that $\phi$ holds on all states of some path. Such invariant behaviour cannot be expressed with a condition such as in (6.12), but is adequately defined by demanding that

$$\rho \vDash \nu Z.f \text{ iff } (\rho \vDash \nu_m Z.f \text{ for all } m \geq 0). \tag{6.14}$$

A dual reasoning to the above shows that $\rho \vDash \nu Z.Z$ holds, regardless of the nature of $\rho$.

One informal way of understanding the definitions in (6.12) and (6.14) is that $\rho \vDash \mu Z.f$ is false until, and if, it is proven to hold; whereas $\rho \vDash \nu Z.f$ is true until, and if, it is proven to be false. The temporal aspect is encoded by the unfolding of the recursion in (6.11), or in (6.13).

To prove that this recursive way of specifying $\rho \vDash f$ actually is well defined, one has to consider more general forms of induction which keep track not only of the height of $f$'s parse tree, but also of the number of syntactic approximants $\mu_m Z.g$ and $\nu_n Z.h$, their 'degree' (in this case, $m$ and $n$), as well as their 'alternation' (the body of a fixed point may contain a free occurrence of a variable for a recursion higher up in the parse tree). This can be done, though we won't discuss the details here.

### 6.4.2 Coding CTL models and specifications

Given a CTL model $\mathcal{M} = (S, \rightarrow, L)$, the $\mu$ and $\nu$ operators permit us to translate any CTL formula $\phi$ into a formula, $f^\phi$, of the relational mu-calculus such that $f^\phi$ represents the set of states $s \in S$ with $s \vDash \phi$. Since we already saw how to represent subsets of states as such formulas, we can then capture

the model-checking problem

$$\mathcal{M}, I \overset{?}{\vDash} \phi \tag{6.15}$$

of whether all *initial* states $s \in I$ satisfy $\phi$, in purely symbolic form: we answer in the affirmative if $f^I \cdot \overline{f}^\phi$ is unsatisfiable, where $f^I$ is the characteristic function of $I \subseteq S$. Otherwise, the logical structure of $f^I \cdot \overline{f}^\phi$ may be exploited to extract debugging information for correcting the model $\mathcal{M}$ in order to make (6.15) true.

Recall how we can represent the transition relation $\rightarrow$ as a boolean formula $f^\rightarrow$ (see Section 6.3.2). As before, we assume that states are coded as bit vectors $(v_1, v_2, \ldots, v_n)$ and so the free boolean variables of all functions $f^\phi$ are subsumed by the vector $\hat{x}$. The coding of the CTL formula $\phi$ as a function $f^\phi$ in the relational mu-calculus is now given inductively as follows:

$$\begin{aligned}
f^x &\overset{\text{def}}{=} x \quad \text{for variables } x \\
f^\perp &\overset{\text{def}}{=} 0 \\
f^{\neg\phi} &\overset{\text{def}}{=} \overline{f^\phi} \\
f^{\phi\wedge\psi} &\overset{\text{def}}{=} f^\phi \cdot f^\psi \\
f^{\mathrm{EX}\,\phi} &\overset{\text{def}}{=} \exists \hat{x}'.\, (f^\rightarrow \cdot f^\phi[\hat{x} := \hat{x}']).
\end{aligned}$$

The clause for EX deserves explanation. The variables $x_i$ refer to the current state, whereas $x_i'$ refer to the next state. The semantics of CTL says that $s \vDash \mathrm{EX}\,\phi$ if, and only if, there is some $s'$ with $s \rightarrow s'$ and $s' \vDash \phi$. The boolean formula encodes this definition, computing 1 precisely when this is the case. If $\hat{x}$ models the current state $s$, then $\hat{x}'$ models a possible successor state if $f^\rightarrow$, a function in $(\hat{x}, \hat{x}')$, holds. We use the nested boolean quantifier $\exists \hat{x}'$ in order to say 'there is *some* successor state.' Observe also the desired effect of $[\hat{x} := \hat{x}']$ performed on $f^\phi$, thereby 'forcing' $\phi$ to be true at some next state[4].

The clause for EF is more complicated and involves the $\mu$ operator. Recall the equivalence

$$\mathrm{EF}\,\phi \equiv \phi \vee \mathrm{EX}\,\mathrm{EF}\,\phi. \tag{6.16}$$

[4] Exercise 6 on page 409 should give you a feel for how the semantics of $f[\hat{x} := \hat{x}']$ does not interfere with potential $\exists \hat{x}'$ or $\forall \hat{x}'$ quantifiers within $f$. For example, to evaluate $\rho \vDash (\exists \hat{x}'.f)[\hat{x} := \hat{x}']$, we evaluate $\rho[\hat{x} := \hat{x}'] \vDash \exists \hat{x}'.f$, which is true if we can find some values $(v_1, v_2, \ldots, v_n) \in \{0,1\}^n$ such that $\rho[\hat{x} := \hat{x}'][x_1' \mapsto v_1][x_2' \mapsto v_2] \ldots [x_n' \mapsto v_n] \vDash f$ is true. Observe that the resulting environment binds all $x_i'$ to $v_i$, but for all other values it binds them according to $\rho[\hat{x} := \hat{x}']$; since the latter binds $x_i$ to $\rho(x_i')$ which is the 'old' value of $x_i'$, this is exactly what we desire in order to prevent a clash of variable names with the intended semantics.
Recall that an OBDD implementation synthesises formulas in a bottom-up fashion, so a reduced OBDD for $\exists \hat{x}'.f$ will not contain any $x_i'$ nodes as its function does not depend on those variables. Thus, OBDDs also avoid such name clash problems.

Therefore, $f^{\mathrm{EF}\,\phi}$ has to be equivalent to $f^{\phi} + f^{\mathrm{EX}\,\mathrm{EF}\,\phi}$ which in turn is equivalent to $f^{\phi} + \exists \hat{x}'.\,(f^{\rightarrow} \cdot f^{\mathrm{EF}\,\phi}[\hat{x} := \hat{x}'])$. Now, since EF involves computing the *least* fixed point of the operator derived from the Equivalence (6.16), we obtain

$$f^{\mathrm{EF}\,\phi} \stackrel{\mathrm{def}}{=} \mu Z.\,(f^{\phi} + \exists \hat{x}'.\,(f^{\rightarrow} \cdot Z[\hat{x} := \hat{x}'])). \tag{6.17}$$

Note that the substitution $Z[\hat{x} := \hat{x}']$ means that the boolean function $Z$ should be made to depend on the $x_i'$ variables, rather than the $x_i$ variables. This is because the evaluation of $\rho \vDash Z[\hat{x} := \hat{x}']$ results in $\rho[\hat{x} := \hat{x}'] \vDash Z$, where the latter valuation satisfies $\rho[\hat{x} := \hat{x}'](x_i) = \rho(x_i')$. Then, we use the modified valuation $\rho[\hat{x} := \hat{x}']$ to evaluate $Z$.

Since $\mathrm{EF}\,\phi$ is equivalent to $\mathrm{E}[\top\ \mathrm{U}\ \phi]$, we can generalise our coding of $\mathrm{EF}\,\phi$ accordingly:

$$f^{\mathrm{E}[\phi \mathrm{U} \psi]} \stackrel{\mathrm{def}}{=} \mu Z.\,(f^{\psi} + f^{\phi} \cdot \exists \hat{x}'.\,(f^{\rightarrow} \cdot Z[\hat{x} := \hat{x}'])). \tag{6.18}$$

The coding of AF is similar to the one for EF in (6.17), except that 'for some' (boolean quantification $\exists \hat{x}'$) gets replaced by 'for all' (boolean quantification $\forall \hat{x}'$) and the 'conjunction' $f^{\rightarrow} \cdot Z[\hat{x} := \hat{x}']$ turns into the 'implication' $\overline{f^{\rightarrow}} + Z[\hat{x} := \hat{x}']$:

$$f^{\mathrm{AF}\,\phi} \stackrel{\mathrm{def}}{=} \mu Z.\,(f^{\phi} + \forall \hat{x}'.\,(\overline{f^{\rightarrow}} + Z[\hat{x} := \hat{x}'])). \tag{6.19}$$

Notice how the semantics of $\mu Z.f$ in (6.12) reflects the intended meaning of the AF connective. The $m$th approximant of $f^{\mathrm{AF}\,\phi}$, which we write as $f_m^{\mathrm{AF}\,\phi}$, represents those states where all paths reach a $\phi$-state within $m$ steps.

This leaves us with coding EG, for then we have provided such a coding for an adequate fragment of CTL (recall Theorem 3.17 on page 216). Because EG involves computing greatest fixed points, we make use of the $\nu$ operator:

$$f^{\mathrm{EG}\,\phi} \stackrel{\mathrm{def}}{=} \nu Z.\,(f^{\phi} \cdot \exists \hat{x}'.\,(f^{\rightarrow} \cdot Z[\hat{x} := \hat{x}'])). \tag{6.20}$$

Observe that this does follow the logical structure of the semantics of EG: we need to show $\phi$ in the present state and then we have to find some successor state satisfying $\mathrm{EG}\,\phi$. The crucial point is that this obligation never ceases; this is exactly what we ensured in (6.14).

Let us see these codings in action on the model of Figure 6.24. We want to perform a symbolic model check of the formula $\mathrm{EX}\,(x_1 \vee \neg x_2)$. You should verify, using e.g. the labelling algorithm from Chapter 3, that $[\![\mathrm{EX}\,(x_1 \vee \neg x_2)]\!] = \{s_1, s_2\}$. Our claim is that this set is computed symbolically by the resulting formula $f^{\mathrm{EX}\,(x_1 \vee \neg x_2)}$. First, we compute the formula

$f^{\rightarrow}$ which represents the transition relation $\rightarrow$:

$$f^{\rightarrow} = (x_1' \leftrightarrow \overline{x}_1 \cdot \overline{x}_2 \cdot u) \cdot (x_2' \leftrightarrow x_1)$$

where $u$ is an input variable used to model the non-determinism (compare the form (6.6) for the transition relation in Section 6.3.4). Thus, we obtain

$$\begin{aligned} f^{\mathrm{EX}\,(x_1 \vee \neg x_2)} &= \exists x_1'.\exists x_2'.(f^{\rightarrow} \cdot f^{x_1 \vee \neg x_2}[\hat{x} := \hat{x}']) \\ &= \exists x_1'.\exists x_2'.((x_1' \leftrightarrow \overline{x}_1 \cdot \overline{x}_2 \cdot u) \cdot (x_2' \leftrightarrow x_1) \cdot (x_1' + \overline{x}_2')). \end{aligned}$$

To see whether $s_0$ satisfies $\mathrm{EX}\,(x_1 \vee \neg x_2)$, we evaluate $\rho_0 \vDash f^{\mathrm{EX}\,(x_1 \vee \neg x_2)}$, where $\rho_0(x_1) = 1$ and $\rho_0(x_2) = 0$ (the value of $\rho_0(u)$ does not matter). We find that this does not hold, whence $s_0 \nvDash \mathrm{EX}\,(x_1 \vee \neg x_2)$. Likewise, we verify $s_1 \vDash \mathrm{EX}\,(x_1 \vee \neg x_2)$ by showing $\rho_1 \vDash f^{\mathrm{EX}\,(x_1 \vee \neg x_2)}$; and $s_2 \vDash \mathrm{EX}\,(x_1 \vee \neg x_2)$ by showing $\rho_2 \vDash f^{\mathrm{EX}\,(x_1 \vee \neg x_2)}$, where $\rho_i$ is the valuation representing state $s_i$.

As a second example, we compute $f^{\mathrm{AF}\,(\neg x_1 \wedge \neg x_2)}$ for the model in Figure 6.24. First, note that all three[5] states satisfy $\mathrm{AF}\,(\neg x_1 \wedge \neg x_2)$, if we apply the labelling algorithm to the explicit model. Let us verify that the symbolic encoding matches this result. By (6.19), we have that $f^{\mathrm{AF}\,(\neg x_1 \wedge \neg x_2)}$ equals

$$\mu Z.\left((\overline{x}_1 \cdot \overline{x}_2) + \forall x_1'.\forall x_2'.(x_1' \leftrightarrow \overline{x}_1 \cdot \overline{x}_2 \cdot u) \cdot (x_2' \leftrightarrow x_1) \cdot Z[\hat{x} := \hat{x}']\right). \quad (6.21)$$

By (6.12), we have $\rho \vDash f^{\mathrm{AF}\,(\neg x_1 \wedge \neg x_2)}$ iff $\rho \vDash f_m^{\mathrm{AF}\,(\neg x_1 \wedge \neg x_2)}$ for some $m \geq 0$. Clearly, we have $\rho \nvDash f_0^{\mathrm{AF}\,(\neg x_1 \wedge \neg x_2)}$. Now, $f_1^{\mathrm{AF}\,(\neg x_1 \wedge \neg x_2)}$ equals

$$((\overline{x}_1 \cdot \overline{x}_2) + \forall x_1'.\forall x_2'.(x_1' \leftrightarrow \overline{x}_1 \cdot \overline{x}_2 \cdot u) \cdot (x_2' \leftrightarrow x_1) \cdot Z[\hat{x} := \hat{x}'])[0/Z].$$

Since $[0/Z]$ is a meta-operation, the latter formula is just

$$(\overline{x}_1 \cdot \overline{x}_2) + \forall x_1'.\forall x_2'.(x_1' \leftrightarrow \overline{x}_1 \cdot \overline{x}_2 \cdot u) \cdot (x_2' \leftrightarrow x_1) \cdot 0[\hat{x} := \hat{x}'].$$

Thus, we need to evaluate the disjunction $(\overline{x}_1 \cdot \overline{x}_2) + \forall x_1'.\forall x_2'.(x_1' \leftrightarrow \overline{x}_1 \cdot \overline{x}_2 \cdot u) \cdot (x_2' \leftrightarrow x_1) \cdot 0[\hat{x} := \hat{x}']$ at $\rho$. In particular, if $\rho(x_1) = 0$ and $\rho(x_2) = 0$, then $\rho \vDash \overline{x}_1 \cdot \overline{x}_2$ and so $\rho \vDash (\overline{x}_1 \cdot \overline{x}_2) + \forall x_1'.\forall x_2'.(x_1' \leftrightarrow \overline{x}_1 \cdot \overline{x}_2 \cdot u) \cdot (x_2' \leftrightarrow x_1) \cdot 0[\hat{x} := \hat{x}']$. Thus, $s_2 \vDash \mathrm{AF}\,(\neg x_1 \wedge \neg x_2)$ holds.

Similar reasoning establishes that the formula in (6.21) renders a correct coding for the remaining two states as well, which you are invited to verify as an exercise.

**Symbolic model checking with fairness** In Chapter 3, we sketched how SMV could use fairness assumptions which were not expressible entirely

[5] Since we have added the variable $u$, there are actually six states; they all satisfy the formula.

within CTL and its semantics. The addition of fairness could be achieved by restricting the ordinary CTL semantics to fair computation paths, or fair states. Formally, we were given a set $C = \{\psi_1, \psi_2, \ldots, \psi_k\}$ of CTL formulas, called the *fairness constraints*, and we wanted to check whether $s \vDash \phi$ holds for a CTL formula $\phi$ and all initial states $s$, with the additional fairness constraints in $C$. Since $\bot$, $\neg$, $\wedge$, EX, EU and EG form an adequate set of connectives for CTL, we may restrict this discussion to only these operators. Clearly, the propositional connectives won't change their meaning with the addition of fairness constraints. Therefore, it suffices to provide symbolic codings for the fair connectives $\mathrm{E}_C\mathrm{X}$, $\mathrm{E}_C\mathrm{U}$ and $\mathrm{E}_C\mathrm{G}$ from Chapter 3. The key is to represent the set of fair states symbolically as a boolean formula $\mathsf{fair}$ defined as

$$\mathsf{fair} \stackrel{\mathrm{def}}{=} f^{\mathrm{E}_C\mathrm{G}\top} \tag{6.22}$$

which uses the (yet to be defined) function $f^{\mathrm{E}_C\mathrm{G}\,\phi}$ with $\top$ as an instance. Assuming that the coding of $f^{\mathrm{E}_C\mathrm{G}\,\phi}$ is correct, we see that $\mathsf{fair}$ computes 1 in a state $s$ if, and only if, there is a fair path with respect to $C$ that begins in $s$. We say that such an $s$ is a *fair state*.

As for $\mathrm{E}_C\mathrm{X}$, note that $s \vDash \mathrm{E}_C\mathrm{X}\phi$ if, and only if, there is some next state $s'$ with $s \rightarrow s'$ and $s' \vDash \phi$ such that $s'$ is a fair state. This immediately renders

$$f^{\mathrm{E}_C\mathrm{X}\phi} \stackrel{\mathrm{def}}{=} \exists \hat{x}'.(f^{\rightarrow} \cdot (f^{\phi} \cdot \mathsf{fair})[\hat{x} := \hat{x}']). \tag{6.23}$$

Similarly, we obtain

$$f^{\mathrm{E}_C[\phi_1 \mathrm{U} \phi_2]} \stackrel{\mathrm{def}}{=} \mu Z.\,(f^{\phi_2} \cdot \mathsf{fair} + f^{\phi_1} \cdot \exists \hat{x}'.\,(f^{\rightarrow} \cdot Z[\hat{x} := \hat{x}'])). \tag{6.24}$$

This leaves us with the task of coding $f^{\mathrm{E}_C\mathrm{G}\,\phi}$. It is this last connective which reveals the complexity of fairness checks at work. Because the coding of $f^{\mathrm{E}_C\mathrm{G}\,\phi}$ is rather complex, we proceed in steps. It is convenient to have the EX and EU functionality also at the level of boolean formulas directly. For example, if $f$ is a boolean function in $\hat{x}$, then `checkEX` $(f)$ codes the boolean formula which computes 1 for those vectors $\hat{x}$ which have a next state $\hat{x}'$ for which $f$ computes 1:

$$\texttt{checkEX}\,(f) \stackrel{\mathrm{def}}{=} \exists \hat{x}'.(f^{\rightarrow} \cdot f[\hat{x} := \hat{x}']). \tag{6.25}$$

Thus, $f^{\mathrm{E}_C\mathrm{X}\phi}$ equals `checkEX` $(f^{\phi} \cdot \mathsf{fair})$. We proceed in the same way for functions $f$ and $g$ in $n$ arguments $\hat{x}$ to obtain `checkEU` $(f, g)$ which computes

1 at $\hat{x}$ if there is a path that realises the $f$ U $g$ pattern:

$$\texttt{checkEU}\,(f, g) \stackrel{\text{def}}{=} \mu Y.g + (f \cdot \texttt{checkEX}(Y)). \tag{6.26}$$

With this in place, we can code $f^{\mathrm{E}_C \mathrm{G}\phi}$ quite easily:

$$f^{\mathrm{E}_C \mathrm{G}\phi} \stackrel{\text{def}}{=} \nu Z.f^{\phi} \cdot \prod_{i=1}^{k} \texttt{checkEX}\,(\texttt{checkEU}\,(f^{\phi}, Z \cdot f^{\psi_i}) \cdot \mathsf{fair}). \tag{6.27}$$

Note that this coding has a least fixed point (`checkEU`) in the body of a greatest fixed point. This is computationally rather involved since the call of `checkEU` contains $Z$, the recursion variable of the outer greatest fixed point, as a free variable; thus these recursions are nested and inter-dependent; the recursions 'alternate.' Observe how this coding operates: to have a fair path from $\hat{x}$ on which $\phi$ holds globally, we need $\phi$ to hold at $\hat{x}$; and for all fairness constraints $\psi_i$ there has to be a next state $\hat{x}'$, where the whole property is true again (enforced by the free $Z$) and each fairness constraint is realised eventually on that path. The recursion in $Z$ constantly reiterates this reasoning, so if this function computes 1, then there is a path on which $\phi$ holds globally and where each $\psi_i$ is true infinitely often.

## 6.5 Exercises

Exercises 6.1

1. Write down the truth tables for the boolean formulas in Example 6.2 on page 359. In your table, you may use 0 and 1, or F and T, whatever you prefer. What truth value does the boolean formula of item (4) on page 359 compute?
2. $\oplus$ is the exclusive-or: $x \oplus y \stackrel{\text{def}}{=} 1$ if the values of $x$ and $y$ are different; otherwise, $x \oplus y \stackrel{\text{def}}{=} 0$. Express this in propositional logic, i.e. find a formula $\phi$ having the same truth table as $\oplus$.
3. * Write down a boolean formula $f(x, y)$ in terms of $\cdot$, $+$, $\bar{\ }$, 0 and 1, such that $f$ has the same truth table as $p \to q$.
4. Write down a BNF for the syntax of boolean formulas based on the operations in Definition 6.1.

---

Exercises 6.2

1. * Suppose we swap all dashed and solid lines in the binary decision tree of Figure 6.2. Write out the truth table of the resulting binary decision tree and find a formula for it.

* 2. Consider the following truth table:

| $p$ | $q$ | $r$ | $\phi$ |
|---|---|---|---|
| T | T | T | T |
| T | T | F | F |
| T | F | T | F |
| T | F | F | F |
| F | T | T | T |
| F | T | F | F |
| F | F | T | T |
| F | F | F | F |

Write down a binary decision tree which represents the boolean function specified in this truth table.

3. Construct a binary decision tree for the boolean function specified in Figure 6.2, but now the root should be a $y$-node and its two successors should be $x$-nodes.
4. Consider the following boolean function given by its truth table:

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |

(a) Construct a binary decision tree for $f(x, y, z)$ such that the root is an $x$-node followed by $y$- and then $z$-nodes.
(b) Construct another binary decision tree for $f(x, y, z)$, but now let its root be a $z$-node followed by $y$- and then $x$-nodes.

5. Let $T$ be a binary decision tree for a boolean function $f(x_1, x_2, \ldots, x_n)$ of $n$ boolean variables. Suppose that every variable occurs exactly once as one travels down on any path of the tree $T$. Use mathematical induction to show that $T$ has $2^{n+1} - 1$ nodes.

---

Exercises 6.3

* 1. Explain why all reductions C1–C3 (page 363) on a BDD $B$ result in BDDs which still represent the same function as $B$.
2. Consider the BDD in Figure 6.7.
 * (a) Specify the truth table for the boolean function $f(x, y, z)$ represented by this BDD.

(b) Find a BDD for that function which does not have multiple occurrences of variables along any path.

3. Let $f$ be the function represented by the BDD of Figure 6.3(b). Using also the BDDs $B_0$, $B_1$ and $B_x$ illustrated in Figure 6.6, find BDDs representing
   (a) $f \cdot x$
   (b) $x + f$
   (c) $\overline{f \cdot 0}$
   (d) $f \cdot 1$.

---

Exercises 6.4

1. Figure 6.9 (page 367) shows a BDD with ordering $[x, y, z]$.
   * (a) Find an equivalent reduced BDD with ordering $[z, y, x]$. (Hint: find first the decision tree with the ordering $[z, y, x]$, and then reduce it using C1–C3.)
   (b) Carry out the same construction process for the variable ordering $[y, z, x]$. Does the reduced BDD have more or fewer nodes than the ones for the orderings $[x, y, z]$ and $[z, y, x]$?
2. Consider the BDDs in Figures 6.4–6.10. Determine which of them are OBDDs. If you find an OBDD, you need to specify a list of its boolean variables without double occurrences which demonstrates that ordering.
3. Consider the following boolean formulas. Compute their unique reduced OBDDs with respect to the ordering $[x, y, z]$. It is advisable to first compute a binary decision tree and then to perform the removal of redundancies.
   (a) $f(x, y) \stackrel{\text{def}}{=} x \cdot y$
   * (b) $f(x, y) \stackrel{\text{def}}{=} x + y$
   (c) $f(x, y) \stackrel{\text{def}}{=} x \oplus y$
   * (d) $f(x, y, z) \stackrel{\text{def}}{=} (x \oplus y) \cdot (\overline{x} + z)$.
4. Recall the derived connective $\phi \leftrightarrow \psi$ from Chapter 1 saying that for all valuations $\phi$ is true if, and only if, $\psi$ is true.
   (a) Define this operator for boolean formulas using the basic operations $\cdot$, $+$, $\oplus$ and $\bar{}$ from Definition 6.1.
   (b) Draw a reduced OBDD for the formula $g(x, y) \stackrel{\text{def}}{=} x \leftrightarrow y$ using the ordering $[y, x]$.
5. Consider the even parity function introduced at the end of the last section.
   (a) Define the odd parity function $f_{\text{odd}}(x_1, x_2, \ldots, x_n)$.
   (b) Draw an OBDD for the odd parity function for $n = 5$ and the ordering $[x_3, x_5, x_1, x_4, x_2]$. Would the overall structure of this OBDD change if you changed the ordering?
   (c) Show that $f_{\text{even}}(x_1, x_2, \ldots, x_n)$ and $\overline{f_{\text{odd}}(x_1, x_2, \ldots, x_n)}$ denote the same boolean function.
6. Use Theorem 6.7 (page 368) to show that, if the reductions C1–C3 are applied until no more reduction is possible, the result is independent of the order in which they were applied.

---

Exercises 6.5

1. Given the boolean formula $f(x_1, x_2, x_3) \stackrel{\text{def}}{=} x_1 \cdot (x_2 + \overline{x_3})$, compute its reduced OBDD for the following orderings:
   (a) $[x_1, x_2, x_3]$
   (b) $[x_3, x_1, x_2]$
   (c) $[x_3, x_2, x_1]$.
2. Compute the reduced OBDD for $f(x, y, z) = x \cdot (z + \overline{z}) + \overline{y} \cdot \overline{x}$ in any ordering you like. Is there a $z$-node in that reduced OBDD?
3. Consider the boolean formula $f(x, y, z) \stackrel{\text{def}}{=} (\overline{x} + y + \overline{z}) \cdot (x + \overline{y} + z) \cdot (x + y)$. For the variable orderings below, compute the (unique) reduced OBDD $B_f$ of $f$ with respect to that ordering. It is best to write down the binary decision tree for that ordering and then to apply all possible reductions.
   (a) $[x, y, z]$.
   (b) $[y, x, z]$.
   (c) $[z, x, y]$.
   (d) Find an ordering of variables for which the resulting reduced OBDD $B_f$ has a minimal number of edges; i.e. there is no ordering for which the corresponding $B_f$ has fewer edges. (How many possible orderings for $x$, $y$ and $z$ are there?)
4. Given the truth table

| $x$ | $y$ | $z$ | $f(x, y, z)$ |
|---|---|---|---|
| 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 |

   compute the reduced OBDD with respect to the following ordering of variables:
   (a) $[x, y, z]$
   (b) $[z, y, x]$
   (c) $[y, z, x]$
   (d) $[x, z, y]$.
5. Given the ordering $[p, q, r]$, compute the reduced BDDs for $p \wedge (q \vee r)$ and $(p \wedge q) \vee (p \wedge r)$ and explain why they are identical.

* 6. Consider the BDD in Figure 6.11 (page 370).
   (a) Construct its truth table.
   (b) Compute its conjunctive normal form.
   (c) Compare the length of that normal form with the size of the BDD. What is your assessment?

Exercises 6.6

1. Perform the execution of `reduce` on the following OBDDs:
   (a) The binary decision tree for
      i. $x \oplus y$
      ii. $x \cdot y$
      iii. $x + y$
      iv. $x \leftrightarrow y$.
   (b) The OBDD in Figure 6.2 (page 361).
   * (c) The OBDD in Figure 6.4 (page 363).

———

Exercises 6.7

1. Recall the Shannon expansion in (6.1) on page 374. Suppose that $x$ does not occur in $f$ at all. Why does (6.1) still hold?
2. Let $f(x, y, z) \stackrel{\text{def}}{=} y + \overline{z} \cdot x + z \cdot \overline{y} + y \cdot x$ be a boolean formula. Compute $f$'s Shannon expansion with respect to
   (a) $x$
   (b) $y$
   (c) $z$.
3. Show that boolean formulas $f$ and $g$ are semantically equivalent if, and only if, the boolean formula $(\overline{f} + g) \cdot (f + \overline{g})$ computes 1 for all possible assignments of 0s and 1s to their variables.
4. We may use the Shannon expansion to define formally how BDDs determine boolean functions. Let $B$ be a BDD. It is intuitively clear that $B$ determines a unique boolean function. Formally, we compute a function $f_n$ inductively (bottom-up) for all nodes $n$ of $B$:
   – If $n$ is a terminal node labelled 0, then $f_n$ is the constant 0 function.
   – Dually, if $n$ is a terminal 1-node, then $f_n$ is the constant 1 function.
   – If $n$ is a non-terminal node labelled $x$, then we already have defined the boolean functions $f_{\mathrm{lo}(n)}$ and $f_{\mathrm{hi}(n)}$ and set $f_n$ to be $\overline{x} \cdot f_{\mathrm{lo}(n)} + x \cdot f_{\mathrm{hi}(n)}$.

   If $i$ is the initial node of $B$, then $f_i$ is the boolean function represented by $B$. Observe that we could apply this definition as a symbolic evaluation of $B$ resulting in a boolean formula. For example, the BDD of Figure 6.3(b) renders $\overline{x} \cdot (\overline{y} \cdot 1 + y \cdot 0) + x \cdot 0$. Compute the boolean formulas obtained in this way for the following BDDs:
   (a) the BDD in Figure 6.5(b) (page 364)
   (b) the BDDs in Figure 6.6 (page 365)
   (c) the BDD in Figure 6.11 (page 370).
* 5. Consider a ternary (= takes three arguments) boolean connective $f \to (g, h)$ which is equivalent to $g$ when $f$ is true; otherwise, it is equivalent to $h$.
   (a) Define this connective using any of the operators $+$, $\cdot$, $\oplus$ or $\overline{\ }$.
   (b) Recall exercise 4. Use the ternary operator above to write $f_n$ as an expression of $f_{\mathrm{lo}(n)}$, $f_{\mathrm{hi}(n)}$ and its label $x$.

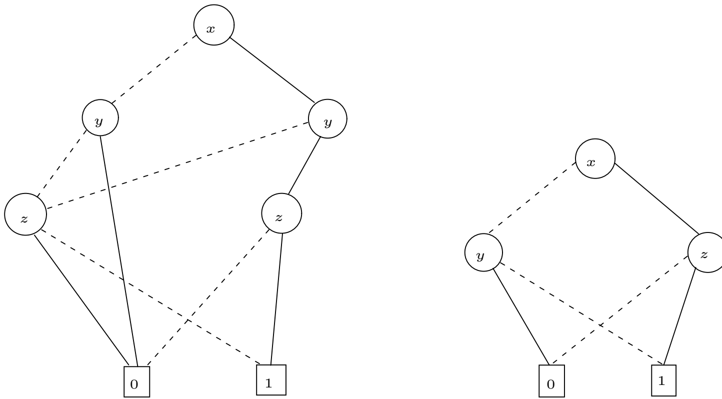


**Figure 6.30.** The reduced OBDDs $B_f$ and $B_g$ (see exercises).

(c) Use mathematical induction (on what?) to prove that, if the root of $f_n$ is an $x$-node, then $f_n$ is independent of any $y$ which comes before $x$ in an assumed variable ordering.

6. Explain why `apply` $(\mathrm{op}, B_f, B_g)$, where $B_f$ and $B_g$ have compatible ordering, produces an OBDD with an ordering compatible with that of $B_f$ and $B_g$.
7. Explain why the four cases of the control structure for `apply` are exhaustive, i.e. there are no other possible cases in its execution.
8. Consider the reduced OBDDs $B_f$ and $B_g$ in Figure 6.30. Recall that, in order to compute the reduced OBDD for $f$ op $g$, you need to
   – construct the tree showing the recursive descent of `apply` $(\mathrm{op}, B_f, B_g)$ as done in Figure 6.16;
   – use that tree to simulate `apply` $(\mathrm{op}, B_f, B_g)$; and
   – reduce, if necessary, the resulting OBDD.

   Perform these steps on the OBDDs of Figure 6.30 for the operation 'op' being
   (a) $+$
   (b) $\oplus$
   (c) $\cdot$
9. Let $B_f$ be the OBDD in Figure 6.11 (page 370). Compute `apply` $(\oplus, B_f, B_1)$ and reduce the resulting OBDD. If you did everything correctly, then this OBDD should be isomorphic to the one obtained from swapping 0- and 1-nodes in Figure 6.11.

* 10. Consider the OBDD $B_c$ in Figure 6.31 which represents the 'don't care' conditions for comparing the boolean functions $f$ and $g$ represented in Figure 6.30. This means that we want to compare whether $f$ and $g$ are equal for all values of variables except those for which $c$ is true (i.e. we 'don't care' when $c$ is true).
   (a) Show that the boolean formula $(\overline{f} \oplus g) + c$ is valid (always computes 1) if, and only if, $f$ and $g$ are equivalent on all values for which $c$ evaluates to 0.

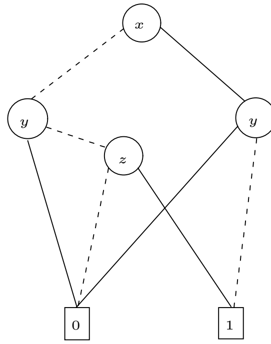


**Figure 6.31.** The reduced OBDD $B_c$ representing the 'don't care' conditions for the equivalence test of the OBDDs in Figure 6.30.

(b) Proceed in three steps as in exercise 8 on page 403 to compute the reduced OBDD for $(\overline{f} \oplus g) + c$ from the OBDDs for $f$, $g$ and $c$. Which call to `apply` needs to be first?

11. We say that $v \in \{0, 1\}$ is a (left)-controlling value for the operation op, if either $v \, \mathrm{op} \, x = 1$ or $v \, \mathrm{op} \, x = 0$ for all values of $x$. We say that $v$ is a controlling value if it is a left- and right-controlling value.
   (a) Define the notion of a right-controlling value.
   (b) Give examples of operations with controlling values.
   (c) Describe informally how `apply` can be optimised when op has a controlling value.
   (d) Could one still do some optimisation if op had only a left- or right-controlling value?
12. We showed that the worst-time complexity of `apply` is $O(|B_f| \cdot |B_g|)$. Show that this upper bound is hard, i.e. it cannot be improved:
   (a) Consider the functions $f(x_1, x_2, \ldots, x_{2n+2m}) \stackrel{\mathrm{def}}{=} x_1 \cdot x_{n+m+1} + \cdots + x_n \cdot x_{2n+m}$ and $g(x_1, x_2, \ldots, x_{2n+2m}) \stackrel{\mathrm{def}}{=} x_{n+1} \cdot x_{2n+m+1} + \cdots + x_{n+m} \cdot x_{2n+2m}$ which are in sum-of-product form. Compute the sum-of-product form of $f + g$.
   (b) Choose the ordering $[x_1, x_2, \ldots, x_{2n+2m}]$ and argue that the OBDDs $B_f$ and $B_g$ have $2^{n+1}$ and $2^{m+1}$ edges, respectively.
   (c) Use the result from part (a) to conclude that $B_{f+g}$ has $2^{n+m+1}$ edges, i.e. $0.5 \cdot |B_f| \cdot |B_g|$.

---

Exercises 6.8

1. Let $f$ be the reduced OBDD represented in Figure 6.5(b) (page 364). Compute the reduced OBDD for the restrictions:
   (a) $f[0/x]$
   * (b) $f[1/x]$

(c) $f[1/y]$

* (d) $f[0/z]$.

* 2. Suppose that we intend to modify the algorithm `restrict` so that it is capable of computing reduced OBDDs for a general composition $f[g/x]$.
   (a) Generalise Equation (6.1) to reflect the intuitive meaning of the operation $[g/x]$.
   (b) What fact about OBDDs causes problems for computing this composition directly?
   (c) How can we compute this composition given the algorithms discussed so far?

3. We define read-1-BDDs as BDDs $B$ where each boolean variable occurs at most once on any evaluation path of $B$. In particular, read-1-BDDs need not possess an ordering on their boolean variables. Clearly, every OBDD is a read-1-BDD; but not every read-1-BDD is an OBDD (see Figure 6.10). In Figure 6.18 we see a BDD which is not a read-1-BDD; the path for $(x, y, z) \Rightarrow (1, 0, 1)$ 'reads' the value of $x$ twice.
   Critically assess the implementation of boolean formulas via OBDDs to see which implementation details could be carried out for read-1-BDDs as well. Which implementation aspects would be problematic?

4. (For those who have had a course on finite automata.) Every boolean function $f$ in $n$ arguments can be viewed as a subset $L_f$ of $\{0,1\}^n$; defined to be the set of all those bit vectors $(v_1, v_2, \ldots, v_n)$ for which $f$ computes 1. Since this is a finite set, $L_f$ is a regular language and has therefore a deterministic finite automaton with a minimal number of states which accepts $L_f$. Can you match some of the OBDD operations with those known for finite automata? How close is the correspondence? (You may have to consider non-reduced OBDDs.)

5. (a) Show that every boolean function in $n$ arguments can be represented as a boolean formula of the grammar

$$f ::= 0 \mid x \mid \overline{f} \mid f_1 + f_2.$$

   (b) Why does this also imply that every such function can be represented by a reduced OBDD in any variable ordering?

6. Use mathematical induction on $n$ to prove that there are exactly $2^{(2^n)}$ many different boolean functions in $n$ arguments.

---

Exercises 6.9

1. Use the `exists` algorithm to compute the OBDDs for
   (a) $\exists x_3.f$, given the OBDD for $f$ in Figure 6.11 (page 370)
   (b) $\forall y.g$, given the OBDD for $g$ in Figure 6.9 (page 367)
   (c) $\exists x_2.\exists x_3.x_1 \cdot y_1 + x_2 \cdot y_2 + x_3 \cdot y_3$.

2. Let $f$ be a boolean function depending on $n$ variables.
   (a) Show:

i. The formula $\exists x.f$ depends on all those variables that $f$ depends upon, except $x$.
ii. If $f$ computes to 1 with respect to a valuation $\rho$, then $\exists x.f$ computes 1 with respect to the same valuation.
iii. If $\exists x.f$ computes to 1 with respect to a valuation $\rho$, then there is a valuation $\rho'$ for $f$ which agrees with $\rho$ for all variables other than $x$ such that $f$ computes to 1 under $\rho'$.

(b) Can the statements above be shown for the function value 0?

3. Let $\phi$ be a boolean formula.
 * (a) Show that $\phi$ is satisfiable if, and only if, $\exists x.\phi$ is satisfiable.
 (b) Show that $\phi$ is valid if, and only if, $\forall x.\phi$ is valid.
 (c) Generalise the two facts above to nested quantifications $\exists \hat{x}$ and $\forall \hat{x}$. (Use induction on the number of quantified variables.)
4. Show that $\forall \hat{x}.f$ and $\overline{\exists \hat{x}.\overline{f}}$ are semantically equivalent. Use induction on the number of arguments in the vector $\hat{x}$.

---

Exercises 6.10
(For those who know about complexity classes.)

1. Show that 3SAT can be reduced to nested existential boolean quantification. Given an instance of 3SAT, we may think of it as a boolean formula $f$ in product-of-sums form $g_1 \cdot g_2 \cdot \cdots \cdot g_n$, where each $g_i$ is of the form $(l_1 + l_2 + l_3)$ with each $l_j$ being a boolean variable or its complementation. For example, $f$ could be $(x + \overline{y} + z) \cdot (x_5 + x + \overline{x}_7) \cdot (\overline{x}_2 + z + x) \cdot (x_4 + \overline{x}_2 + \overline{x}_4)$.
 (a) Show that you can represent each function $g_i$ with an OBDD of no more than three non-terminals, independently of the chosen ordering.
 (b) Introduce $n$ new boolean variables $z_1, z_2, \ldots, z_n$. We write $\sum_{1 \le i \le n} f_i$ for the expression $f_1 + f_2 + \cdots + f_n$ and $\prod_{1 \le i \le n} f_i$ for $f_1 \cdot f_2 \cdot \cdots \cdot f_n$. Consider the boolean formula $h$, defined as

$$\sum_{1 \le i \le n} \left( \overline{g}_i \cdot z_i \cdot \prod_{1 \le j < i} \overline{z}_j \right). \tag{6.28}$$

 Choose any ordering of variables whose list begins as in $[z_1, z_2, \ldots, z_n, \ldots]$. Draw the OBDD for $h$ (draw only the root nodes for $\overline{g}_i$).
 (c) Argue that the OBDD above has at most $4n$ non-terminal nodes.
 (d) Show that $f$ is satisfiable if, and only if, the OBDD for $\exists z_1.\exists z_2.\ldots.\exists z_n.h$ is not equal to $B_1$.
 (e) Explain why the last item shows a reduction of 3SAT to nested existential quantification.
2. Show that the problem of finding an optimal ordering for representing boolean functions as OBDDs is in coNP.

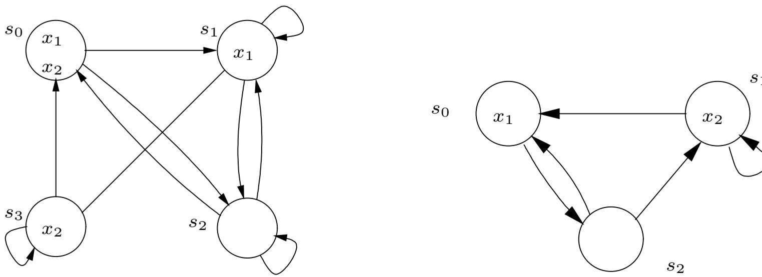


**Figure 6.32.** (a) A CTL model with four states. (b) A CTL model with three states.

3. Recall that $\exists x.f$ is defined as $f[1/x] + f[0/x]$. Since we have efficient algorithms for restriction and $+$, we obtain hereby an efficient algorithm for $\exists z_1. \dots \exists z_n.f$. Thus, P equals NP! What is wrong with this argument?

---

Exercises 6.11

* 1. Consider the CTL model in Figure 6.32(a). Using the ordering $[x_1, x_2]$, draw the OBDD for the subsets $\{s_0, s_1\}$ and $\{s_0, s_2\}$.
2. Consider the CTL model in Figure 6.32(b). Because the number of states is not an exact power of 2, there are more than one OBDDs representing any given set of states. Using again the ordering $[x_1, x_2]$, draw all possible OBDDs for the subsets $\{s_0, s_1\}$ and $\{s_0, s_2\}$.

---

Exercises 6.12

1. Consider the CTL model in Figure 6.32(a).
   (a) Work out the truth table for the transition relation, ordering the columns $[x_1, x_1', x_2, x_2']$. There should be as many 1s in the final column as there are arrows in the transition relation. There is no freedom in the representation in this case, since the number of states is an exact power of 2.
   (b) Draw the OBDD for this transition relation, using the variable ordering $[x_1, x_1', x_2, x_2']$.
2. Apply the algorithm of Section 3.6.1, but now interpreted over OBDDs in the ordering $[x_1, x_2]$, to compute the set of states of the CTL model in Figure 6.32(b) which satisfy
   (a) $\mathrm{AG}\,(x_1 \lor \neg x_2)$
   (b) $\mathrm{E}[x_2 \ \mathrm{U}\ x_1]$.

   Show the OBDDs which are computed along the way.
3. Explain why $\mathtt{exists}(\hat{x}', \mathtt{apply}(\cdot, B_{\to}, B_{X'}))$ faithfully implements the meaning of $\mathrm{pre}_{\exists}(X)$.

---

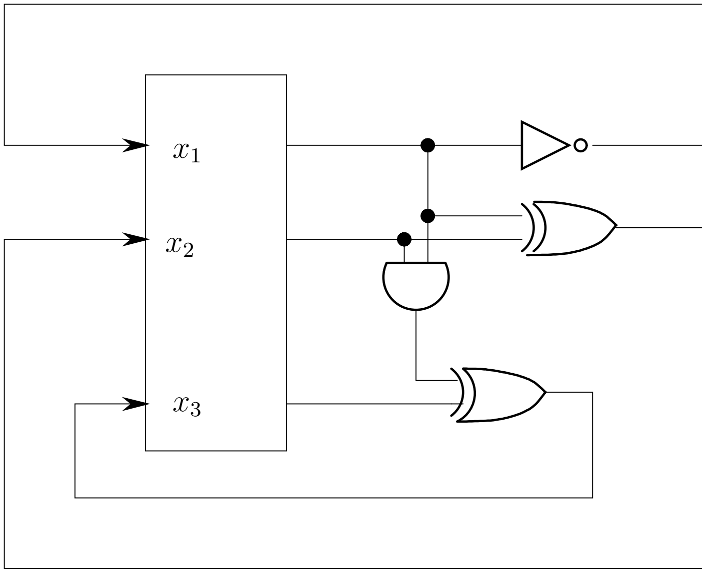


**Figure 6.33.** A synchronous circuit for a modulo 8 counter.

Exercises 6.13

1. (a) Simulate the evolution of the circuit in Figure 6.29 (page 389) with initial state 01. What do you think that it computes?
   (b) Write down the explicit CTL model $(S, \rightarrow, L)$ for this circuit.
2. Consider the sequential synchronous circuit in Figure 6.33.
   (a) Construct the functions $f_i$ for $i = 1, 2, 3$.
   (b) Code the function $f^{\rightarrow}$.
   (c) Recall from Chapter 2 that $(\exists x.\phi) \wedge \psi$ is semantically equivalent to $\exists x.(\phi \wedge \psi)$ if $x$ is not free in $\psi$.
      i. Why is this also true in our setting of boolean formulas?
      ii. Apply this law to push the $\exists$ quantifications in $f^{\rightarrow}$ as far inwards as possible. This is an often useful optimisation in checking synchronous circuits.
3. Consider the boolean formula for the 2-bit comparator:

$$f(x_1, x_2, y_1, y_2) \stackrel{\text{def}}{=} (x_1 \leftrightarrow y_1) \cdot (x_2 \leftrightarrow y_2).$$

   (a) Draw its OBDD for the ordering $[x_1, y_1, x_2, y_2]$.
   (b) Draw its OBDD for the ordering $[x_1, x_2, y_1, y_2]$ and compare that with the one above.
4. (a) Can you use (6.6) from page 388 to code the transition relation $\rightarrow$ of the model in Figure 6.24 on page 384?
   (b) Can you do it with equation (6.9) from page 390?
   (c) With equation (6.8) from page 389?

Exercises 6.14

1. Let $\rho$ be the valuation for which $(x, y, z) \Rightarrow (0, 1, 1)$. Compute whether $\rho \vDash f$ holds for the following boolean formulas:
   (a) $x \cdot (y + \overline{z} \cdot (y \oplus x))$
   (b) $\exists x.(y \cdot (x + z + \overline{y}) + x \cdot \overline{y})$
   (c) $\forall x.(y \cdot (x + z + \overline{y}) + x \cdot \overline{y})$
   (d) $\exists z.(x \cdot \overline{z} + \forall x.((y + (x + \overline{x}) \cdot z)))$
   * (e) $\forall x.(y + \overline{z})$.

* 2. Use (6.14) from page 393 and the definition of the satisfaction relation for formulas of the relational mu-calculus to prove $\rho \vDash \nu Z.Z$ for all valuations $\rho$. In this case, $f$ equals $Z$ and you need to show (6.14) by mathematical induction on $m \geq 0$.

3. An implementation which decides $\vDash$ and $\nvDash$ for the relational mu-calculus obviously cannot represent valuations which assign semantic values 0 or 1 to all, i.e. infinitely many variables. Thus, it makes sense to consider $\vDash$ as a relation between pairs $(\rho, f)$, where $\rho$ only assigns semantic values to all free variables of $f$.
   (a) Assume that $\nu Z$ and $\mu Z$, $\exists x$, $\forall x$, and $[\hat{x} := \hat{x}']$ are binding constructs similar to the quantifiers in predicate logic. Define formally the set of free variables for a formula $f$ of the relational mu-calculus. (Hint: You should define this by structural induction on $f$. Also, which variables get bound in $f[\hat{x} := \hat{x}']$?)
   (b) Recall the notion of $t$ being free for $x$ in $\phi$ which we discussed in Section 2.2.4 Define what '$g$ is free for $Z$ in $f$' should mean and find an example, where $g$ is not free for $Z$ in $f$.
   (c) Explain informally why we can decide whether $\rho \vDash f$ holds, provided that $\rho$ assigns values 0 or 1 to all free variables of $f$. Explain why this answer will be independent of what $\rho$ does to variables which are bound in $f$. Why is this relevant for an implementation framework?

4. Let $\rho$ be the valuation for which $(x, x', y, y') \Rightarrow (0, 1, 1, 1)$. Determine whether $\rho \vDash f$ holds for the following formulas $f$ (recall that we write $f \leftrightarrow g$ as an abbreviation for $\overline{f} \oplus g$, meaning that $f$ computes 1 iff $g$ computes 1):
   (a) $\exists x.(x' \leftrightarrow (\overline{y} + y' \cdot x))$
   (b) $\forall x.(x' \leftrightarrow (\overline{y} + y' \cdot x))$
   (c) $\exists x'.(x' \leftrightarrow (\overline{y} + y' \cdot x))$
   (d) $\forall x'.(x' \leftrightarrow (\overline{y} + y' \cdot x))$.

5. Let $\rho$ be a valuation with $\rho(x_1') = 1$ and $\rho(x_2') = 0$. Determine whether $\rho \vDash f$ holds for the following:
   (a) $\overline{x}_1[\hat{x} := \hat{x}']$
   (b) $(x_1 + \overline{x}_2)[\hat{x} := \hat{x}']$
   (c) $(\overline{x}_1 \cdot \overline{x}_2)[\hat{x} := \hat{x}']$.

6. Evaluate $\rho \vDash (\exists x_1.(x_1 + \overline{x}_2))[\hat{x} := \hat{x}']$ and explain how the valuation $\rho$ changes in that process. In particular, $[\hat{x} := \hat{x}']$ replaces $x_i$ by $x_i'$, but why does this not interfere with the binding quantifier $\exists x_1$?

7. (a) How would you define the notion of semantic entailment for the relational mu-calculus?
   (b) Define formally when two formulas of the relational mu-calculus are semantically equivalent.

---

Exercises 6.15

1. Using the model of Figure 6.24 (page 384), determine whether $\rho \vDash f^{\mathrm{EX}\,(x_1 \vee \neg x_2)}$ holds, where $\rho$ is
   (a) $(x_1, x_2) \Rightarrow (1, 0)$
   (b) $(x_1, x_2) \Rightarrow (0, 1)$
   (c) $(x_1, x_2) \Rightarrow (0, 0)$.
2. Let $S$ be $\{s_0, s_1\}$, with $s_0 \to s_0$, $s_0 \to s_1$ and $s_1 \to s_0$ as possible transitions and $L(s_0) = \{x_1\}$ and $L(s_1) = \emptyset$. Compute the boolean function $f^{\mathrm{EX}\,(\mathrm{EX}\,\neg x_1)}$.
3. Equations (6.17) (page 395), (6.19) and (6.20) define $f^{\mathrm{EF}\,\phi}$, $f^{\mathrm{AF}\,\phi}$ and $f^{\mathrm{EG}\,\phi}$. Write down a similar equation to define $f^{\mathrm{AG}\,\phi}$.
4. Define a direct coding $f^{\mathrm{AU}\,\phi}$ by modifying (6.18) appropriately.
5. Mimic the example checks on page 396 for the connective AU: consider the model of Figure 6.24 (page 384). Since $[\![\mathrm{E}[(x_1 \vee x_2)\ \mathrm{U}\ (\neg x_1 \wedge \neg x_2)]]\!]$ equals the entire state set $\{s_0, s_1, s_2\}$, your coding of $f^{\mathrm{E}[x_1 \vee x_2 \mathrm{U} \neg x_1 \wedge \neg x_2]}$ is correct if it computes 1 for all bit vectors different from $(1, 1)$.
   (a) Verify that your coding is indeed correct.
   (b) Find a boolean formula without fixed points which is semantically equivalent to $f^{\mathrm{E}[(x_1 \vee x_2) \mathrm{U} (\neg x_1 \wedge \neg x_2)]}$.
6. (a) Use (6.20) on page 395 to compute $f^{\mathrm{EG}\,\neg x_1}$ for the model in Figure 6.24.
   (b) Show that $f^{\mathrm{EG}\,\neg x_1}$ faithfully models the set of all states which satisfy $\mathrm{EG}\,\neg x_1$.
7. In the grammar (6.10) for the relational mu-calculus on page 390, it was stated that, in the formulas $\mu Z.f$ and $\nu Z.f$, any occurrence of $Z$ in $f$ is required to fall within an even number of complementation symbols $\bar{\ }$. What happens if we drop this requirement?
   (a) Consider the expression $\mu Z.\overline{Z}$. We already saw that our relation $\rho$ is total in the sense that either $\rho \vDash f$ or $\rho \nvDash f$ holds for all choices of valuations $\rho$ and relational mu-calculus formulas $f$. But formulas like $\mu Z.\overline{Z}$ are not formally monotone. Let $\rho$ be any valuation. Use mutual mathematical induction to show:
      i. $\rho \nvDash \mu_m Z.\overline{Z}$ for all even numbers $m \geq 0$
      ii. $\rho \vDash \mu_m Z.\overline{Z}$ for all odd numbers $m \geq 1$

      Infer from these two items that $\rho \vDash \mu Z.\overline{Z}$ holds according to (6.12).
   (b) Consider any environment $\rho$. Use mathematical induction on $m$ (and maybe an analysis on $\rho$) to show:

      If $\rho \vDash \mu_m Z.(x_1 + \overline{x_2 \cdot \overline{Z}})$ for some $m \geq 0$, then $\rho \vDash \mu_k Z.(x_1 + \overline{x_2 \cdot \overline{Z}})$ for all $k \geq m$.

(c) In general, if $f$ is formally monotone in $Z$ then $\rho \vDash \mu_m Z.f$ implies $\rho \vDash \mu_{m+1} Z.f$. Can you state a similar property for the greatest fixed-point operator $\nu$?

8. Given the CTL model for the circuit in Figure 6.29 (page 389):
   * (a) code the function $f^{\mathrm{EX}\,(x_1 \wedge \neg x_2)}$
   (b) code the function $f^{\mathrm{AG}\,(\mathrm{AF}\,\neg x_1 \wedge \neg x_2)}$
   * (c) find a boolean formula without any fixed points which is semantically equivalent to $f^{\mathrm{AG}\,(\mathrm{AF}\,\neg x_1 \wedge \neg x_2)}$.
9. Consider the sequential synchronous circuit in Figure 6.33 (page 408). Evaluate $\rho \vDash f^{\mathrm{EX}\,x_2}$, where $\rho$ equals
   (a) $(x_1, x_2, x_3) \Rightarrow (1, 0, 1)$
   (b) $(x_1, x_2, x_3) \Rightarrow (0, 1, 0)$.
10. Prove

    **Theorem 6.19** Given a coding for a finite CTL model, let $\phi$ be a CTL formula from an adequate fragment. Then $[\![\phi]\!]$ corresponds to the set of valuations $\rho$ such that $\rho \vDash f^{\phi}$.

    by structural induction on $\phi$. You may first want to show that the evaluation of $\rho \vDash f^{\phi}$ depends only on the values $\rho(x_i)$, i.e. it does not matter what $\rho$ assigns to $x'_i$ or $Z$.
11. Argue that Theorem 6.19 above remains valid for arbitrary CTL formulas as long as we translate formulas $\phi$ which are not in the adequate fragment into semantically equivalent formulas $\psi$ in that fragment and define $f^{\phi}$ to be $f^{\psi}$.
12. Derive the formula $f^{\mathrm{AF}\,(\neg x_1 \wedge x_2)}$ for the model in Figure 6.32(b) on page 407 and evaluate it for the valuation corresponding to state $s_2$ to determine whether $s_2 \vDash \mathrm{AF}\,(\neg x_1 \wedge x_2)$ holds.
13. Repeat the last exercise with $f^{\mathrm{E}[x_1 \vee \neg x_2 \mathrm{U} x_1]}$.
14. Recall the way the two labelling algorithms operate in Chapter 3. Does our symbolic coding mimic either or both of them, or neither?

---

Exercises 6.16

1. Consider the equations in (6.22) and (6.27). The former defines fair in terms of $f^{\mathrm{E}_C\mathrm{G}\top}$, whereas the latter defines $f^{\mathrm{E}_C\mathrm{G}\,\phi}$ for general $\phi$. Why is this unproblematic, i.e. non-circular?
2. Given a fixed CTL model $\mathcal{M} = (S, \rightarrow, L)$, we saw how to code formulas $f^{\phi}$ representing the set of states $s \in S$ with $s \vDash \phi$, $\phi$ being a CTL formula of an adequate fragment.
   (a) Assume the coding without consideration of simple fairness constraints. Use structural induction on the CTL formula $\phi$ to show that
      i. the free variables of $f^{\phi}$ are among $\hat{x}$, where the latter is the vector of boolean variables which code states $s \in S$;
      ii. all fixed-point subformulas of $f^{\phi}$ are formally monotone.

(b) Show these two assertions if $f^{\phi}$ also encodes simple fairness constraints.

3. Consider the pseudo-code for the function `SAT` on page 227. We now want to modify it so that the resulting output is not a set, or an OBDD, but a formula of the relational mu-calculus; thus, we complete the table in Figure 6.22 on page 380 to give formulas of the relational mu-calculus. For example, the output for $\top$ would be 1 and the output for $\mathrm{EU}\,\psi$ would be a recursive call to `SAT` informed by (6.18). Do you have a need for a separate function which handles least or greatest fixed points?
4. (a) Write pseudo-code for a function $\mathtt{SAT}_{\mathrm{rel_mu}}$ which takes as input a formula of the relational mu-calculus, $f$, and synthesises an OBDD $B_f$, representing $f$. Assume that there are no fixed-point subexpressions of $f$ such that their recursive body contains a recursion variable of an outwards fixed point. Thus, the formula in (6.27) is not allowed. The fixed-point operators $\mu$ and $\nu$ require separate subfunctions which iterate the fixed-point meaning informed by (6.12), respectively (6.14). Some of your clauses may need further comment. E.g. how do you handle the constructor $[\hat{x} := \hat{x}']$?
   (b) Explain what goes wrong if the input to your code is the formula in (6.27).
5. If $f$ is a formula with a vector of $n$ free boolean variables $\hat{x}$, then the iteration of $\mu Z.f$, whether as OBDD implementation, or as in (6.12), may require up to $2^n$ recursive unfoldings to compute its meaning. Clearly, this is unacceptable. Given the symbolic encoding of a CTL model $\mathcal{M} = (S, \rightarrow, L)$ and a set $I \subseteq S$ of initial states, we seek a formula that represents all states which are reachable from $I$ on some finite computation path in $\mathcal{M}$. Using the extended Until operator in (6.26), we may express this as `checkEU` $(f^I, \top)$, where $f^I$ is the characteristic function of $I$. We can 'speed up' this iterative process with a technique called 'iterative squaring':

$$\mu Y.(f^{\rightarrow} + \exists \hat{w}.(Y[\hat{x}' := \hat{w}] \cdot Y[\hat{x} := \hat{w}])). \tag{6.29}$$

   Note that this formula depends on the same boolean variables as $f^{\rightarrow}$, i.e. the pair $(\hat{x}, \hat{x}')$. Explain informally:

   > If we apply (6.12) $m$ times to the formula in (6.29), then this has the same semantic 'effect' as applying this rule $2^m$ times to `checkEU` $(f^{\rightarrow}, \top)$.

   Thus, one may first compute the set of states reachable from any initial state and then restrict model checking to those states. Note that this reduction does not alter the semantics of $s \vDash \phi$ for initial states $s$, so it is a sound technique; it sometimes improves, other times worsens, the performance of symbolic model checks.

---

## 6.6 Bibliographic notes

Ordered binary decision diagrams are due to R. E. Bryant [Bry86]. Binary decision diagrams were introduced by C. Y. Lee [Lee59] and S. B. Akers [Ake78]. For a nice survey of these ideas see [Bry92]. For the limitations of OBDDs as models for integer multiplication as well as interesting connections to VLSI design see [Bry91]. A general introduction to the topic of computational complexity and its tight connections to logic can be found in [Pap94]. The modal mu-calculus was invented by D. Kozen [Koz83]; for more on that logic and its application to specifications and verification see [Bra91].

The use of BDDs in model checking was proposed by the team of authors J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill and J. Hwang [BCM+90, CGL93, McM93].

# Bibliography


Ake78. S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, 1978.

AO91. K. R. Apt and E.-R. Olderog. *Verification of Sequential and Concurrent Programs*. Springer-Verlag, 1991.

Bac86. R. C. Backhouse. *Program Construction and Verification*. Prentice Hall, 1986.

BCCZ99. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proceedings of Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99)*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207, 1999.

BCM+90. J. R. Burch, J. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: $10^{20}$ states and beyond. In *IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1990.

BEKV94. K. Broda, S. Eisenbach, H. Khoshnevisan, and S. Vickers. *Reasoned Programming*. Prentice Hall, 1994.

BJ80. G. Boolos and R. Jeffrey. *Computability and Logic*. Cambridge University Press, 2nd edition, 1980.

Boo54. G. Boole. *An Investigation of the Laws of Thought*. Dover, New York, 1854.

Bra91. J. C. Bradfield. *Verifying Temporal Properties of Systems*. Birkhäuser, Boston, 1991.

Bry86. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Compilers*, C-35(8), 1986.

Bry91. R. E. Bryant. On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Applications to Integer Multiplication. *IEEE Transactions on Computers*, 40(2):205–213, February 1991.

Bry92. R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary-decision Diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

CE81. E. M. Clarke and E. A. Emerson. Synthesis of synchronization skeletons for branching time temporal logic. In D. Kozen, editor, *Logic of Programs Workshop*, number 131 in LNCS. Springer Verlag, 1981.

CGL93. E. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In *A Decade of Concurrency*, number 803 in Lecture Notes in Computer Science, pages 124–175. Springer Verlag, 1993.

CGL94. E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and Abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, September 1994.

CGP99. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* MIT Press, 1999.

Che80. B. F. Chellas. *Modal Logic – an Introduction.* Cambridge University Press, 1980.

Dam96. D. R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking.* PhD thesis, Institute for Programming Research and Algorithmics. Eindhoven University of Technology, July 1996.

Dij76. E. W. Dijkstra. *A Discipline of Programming.* Prentice Hall, 1976.

DP96. R. Davies and F. Pfenning. A Modal Analysis of Staged Computation. In *23rd Annual ACM Symposium on Principles of Programming Languages.* ACM Press, January 1996.

EJC03. S. Eisenbach, V. Jurisic, and C. Sadler. Modeling the evolution of .NET programs. In *IFIP International Conference on Formal Methods for Open Distributed Systems*, LNCS. Springer Verlag, 2003.

EN94. R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems.* Benjamin/Cummings, 1994.

FHMV95. R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge.* MIT Press, Cambridge, 1995.

Fit93. M. Fitting. Basic modal logic. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic in Artificial Intelligence and Logic Programming*, volume 1. Oxford University Press, 1993.

Fit96. M. Fitting. *First-Order Logic and Automated Theorem Proving.* Springer, 2nd edition, 1996.

Fra92. N. Francez. *Program Verification.* Addison-Wesley, 1992.

Fre03. G. Frege. *Grundgesetze der Arithmetik, begriffsschriftlich abgeleitet.* 1903. Volumes I and II (Jena).

Gal87. J. H. Gallier. *Logic for Computer Science.* John Wiley, 1987.

Gen69. G. Gentzen. Investigations into logical deduction. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, chapter 3, pages 68–129. North-Holland Publishing Company, 1969.

Gol87. R. Goldblatt. *Logics of Time and Computation.* CSLI Lecture Notes, 1987.

Gri82. D. Gries. A note on a standard strategy for developing loop invariants and loops. *Science of Computer Programming*, 2:207–214, 1982.

Ham78. A. G. Hamilton. *Logic for Mathematicians.* Cambridge University Press, 1978.

Hoa69. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–580, 1969.

Hod77. W. Hodges. *Logic.* Penguin Books, 1977.

Hod83. W. Hodges. Elementary predicate logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 1. Dordrecht: D. Reidel, 1983.

Hol90. G. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, 1990.

JSS01. D. Jackson, I. Shlyakhter, and M. Sridharan. A Micromodularity Mechanism. In *Proceedings of the ACM SIGSOFT Conference on the Foundations of Software Engineering/European Software Engineering Conference (FSE/ESEC'01)*, September 2001.

Koz83. D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

Lee59. C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell System Technical Journal*, 38:985–999, 1959.

Lon83. D. E. Long. *Model Checking, Abstraction, and Compositional Verification.* PhD thesis, School of Computer Science, Carnegie Mellon University, July 1983.

Mar01. A. Martin. Adequate sets of temporal connectives in CTL. *Electronic Notes in Theoretical Computer Science* 52(1), 2001.

McM93. K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

MP91. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1991.

MP95. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, 1995.

MvdH95. J.-J. Ch. Meyer and W. van der Hoek. *Epistemic Logic for AI and Computer Science*, volume 41 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1995.

Pap94. C. H. Papadimitriou. *Computational Complexity.* Addison Wesley, 1994.

Pau91. L.C. Paulson. *ML for the Working Programmer.* Cambridge University Press, 1991.

Pnu81. A. Pnueli. A temporal logic of programs. *Theoretical Computer Science*, 13:45–60, 1981.

Pop94. S. Popkorn. *First Steps in Modal Logic.* Cambridge University Press, 1994.

Pra65. D. Prawitz. *Natural Deduction: A Proof-Theoretical Study.* Almqvist & Wiksell, 1965.

QS81. J. P. Quielle and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of the Fifth International Symposium on Programming*, 1981.

Ros97. A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1997.

SA91. V. Sperschneider and G. Antoniou. *Logic, A Foundation for Computer Science.* Addison Wesley, 1991.

Sch92. U. Schoening. *Logik für Informatiker.* B. I. Wissenschaftsverlag, 1992.

Sch94. D. A. Schmidt. *The Structure of Typed Programming Languages.* Foundations of Computing. The MIT Press, 1994.

Sim94. A. K. Simpson. *The Proof Theory and Semantics of Intuitionistic Modal Logic.* PhD thesis, The University of Edinburgh, Department of Computer Science, 1994.

SS90. G. Stålmarck and M. Såflund. Modeling and verifying systems and software in propositional logic. In B. K. Daniels, editor, *Safety of Computer Control Systems (SAFECOMP'90)*, pages 31–36. Pergamon Press, 1990.

Tay98. R. G. Taylor. *Models of Computation and Formal Languages*. Oxford University Press, 1998.

Ten91. R. D. Tennent. *Semantics of Programming Languages*. Prentice Hall, 1991.

Tur91. R. Turner. *Constructive Foundations for Functional Languages*. McGraw Hill, 1991.

vD89. D. van Dalen. *Logic and Structure*. Universitext. Springer-Verlag, 3rd edition, 1989.

VW84. M. Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. In *Proc. 16th ACM Symposium on Theory of Computing*, pages 446–456, 1984.

Wei98. M. A. Weiss. *Data Structures and Problem Solving Using Java*. Addison-Wesley, 1998.

# Index


ABP, 203
  acknowledgement channel, 203
  alternating the control bit, 203
  fairness, 203
  main SMV program, 207
absorption laws, 88
abstract data type
  sets, 226
abstraction, 175, 229, 247
  and non-determinism, 191
accessibility relation, 309, 320, 336
adequate set of connectives
  for CTL, 216, 222, 231, 397
  for LTL, 186
  for propositional logic, 69, 87, 91
agent, 307, 319, 327
algebraic specification, 170
algorithm
  deterministic, 59
algorithm `apply`, 373
  complexity, 380
  control structure, 374
  recursive descent, 375
algorithm `CNF`, 59
algorithm `reduce`, 372
  complexity, 380
algorithm `restrict`, 377
  complexity, 380
algorithm `reduce`
  example execution, 373
Alloy
  `[]`, 153
  `fun`-statement, 155
  `with`, 146
  assertion, 144
  check directive, 144
  consistency check, 144
  constrain signature, 150
  counterexample, 144
  dot operator, 144
  extending signature, 169
  implication, 147
  let-expression, 153
  limitations, 156
  modifiers, 150
  module, 146
  opening module, 156
  polymorphism, 156
  postcondition, 151
  precondition, 151
  reflexive, transitive closure, 169
  run directive, 146
  signature, 143
    instance, 144
  small scope hypothesis, 143, 168
  transitive closure, 155
  universal quantification, 146
alternating bit protocol, 203
always in the future, 318
and-elimination, 6, 339
and-introduction, 6, 339
application domain, 173, 257
approach
  model-based, 173
  proof-based, 173
approximants
  $\mu_m Z.f$, 392
  $\nu_m Z.f$, 393
arity, 99
array, 299
  bounds, 136, 287
  field, 287
  of integers, 287
  section, 287
artificial intelligence, 306
artificial language, 93
assertion checking, 143
assignment, 191
  initial, 292
  non-deterministic, 205, 230
  program notation, 261
  statement, 261
associativity laws, 56, 88
assumption, 4
  discharging, 28, 329
  temporary, 11, 121

asynchronous
  circuit, 358
  interleaving, 189
atomic formula, 335
  of modal logic, 307
  of predicate logic
    meaning, 124
axiom
  **5**, 331
  **T**, 341
  **4**, 327, 330, 339
  **5**, 330, 339
  **T**, 327, 330, 339, 343
  for assignment, 270
  for equality, 107
  instance, 271

Backus Naur form (BNF), 33
backwards breadth-first search, 225, 232
base case, 41, 42, 86
basic modal logic, 307
BDD, 364
  hi($n$), 372
  lo($n$), 372
  as boolean function, 365
  complement, 367
  consistent path, 365
  edge, 361
  examples, 365
  has an ordering, 367
  layer of variables, 361
  line
    dashed, 362, 365
    solid, 362, 365
  ordered, 367
  read-1, 405
  reduced, 365
  removal of duplicate non-terminals, 363
  removal of duplicate terminals, 363
  removal of redundant tests, 363
  satisfiable, 365
  subBDD, 363
  which is not a read-1-BDD, 378
  which is not an OBDD, 368
  with duplicated subBDDs, 363
belief, 319
binary decision diagram, 364
binary decision tree, 361
  redundancies in, 362
binding priorities, 176, 209
  for basic modal logic, 307
  for integer expressions, 260
  for KT45$^n$, 335
  for predicate logic, 101
  for propositional logic, 5
  for relational mu-calculus, 391
bit, 133
  control, 203
  least significant, 381
  most significant, 381
  one-bit channel, 205
  two-bit channel, 205
blocks of code, 261
Boole, G., 91, 374
boolean algebra, 19
boolean connective, 210, 310
boolean existential quantification, 377
boolean expression, 260, 272
boolean forall quantification, 379
boolean formula
  independent of a variable, 375
  semantically equivalent, 374
  truth table, 359
boolean function
  'don't care' conditions, 403
  as a binary decision tree, 361
  symbolic representation, 358
boolean guard, 282
boolean variable, 358
bottom-elimination, 21
bottom-introduction (see 'not-elimination'), 21
box-elimination, 329
box-introduction, 329
branching-time logic, 174

case
  overlap, 61
case analysis, 61, 62, 113
case-statement, 18, 192
characteristic function, 383
Church, A., 133
circuit
  2-bit comparator, 408
  asynchronous, 194, 389
  sequential, 359
  synchronous, 194, 358, 388, 411
circular definition, 217
Clarke, E., 254
classical logic, 30, 328
client, 259
clock tick, 190
CNF, 55
code
  specification, 257
  verification, 257
coding
  AF, 395
  EF, 394
  EG, 395
  EU, 395
  EX, 394
  examples of symbolic evaluation, 395
  fair EG, 398
  fair EU, 397
  fair EX, 397
  set of fair states, 397
command, 261
  atomic, 261
  compound, 261
common knowledge, 332, 335
  as invariant, 390

communicating processes, 256
communication protocol, 194
Compactness Theorem, 137
completeness
  of natural deduction for predicate logic, 96
  of natural deduction for propositional logic, 54
complexity
  exponential, 229
  of `apply`, 404
  of brute force minimal-sum section algorithm, 288
  of fairness, 397
  of labelling algorithm, 224, 225
  of labelling $\mathrm{EG}_C$, 232
composition
  sequential, 278
  synchronous, 194
compositional semantics, 39
compositionality
  in model checking, 230
computability, 131
computation
  intractable, 49
computation path, 180, 212
  fair, 231
computation trace, 285
computation tree logic, 175
computation tree logic, 306, 318
computational behaviour, 306
computer program, 103
concatenation, 126, 132
conclusion, 4, 273
concurrency, 257
conjunct, 56
conjunction, 4, 291
  infinite, 335
connective
  adequate set, 226
  unary, 177, 209
consistency, 308, 316
consistency checking, 143
constant symbol, 157
constraints
  inconsistent, 74
  SAT solver, 69
contradiction, 20, 118, 319, 325
control structure, 261, 262
controlling value, 404
copy rule, 20, 329
core programming language, 259, 287
correspondence theory, 325
counter example, 123, 130, 317, 354
counter trace, 174
critical section, 187
CTL, 175, 254, 306, 318
  as a subset of CTL*, 218
  expressive power, 220
  modalities, 306
  with boolean combinations of path formulas, 220, 251
CTL connectives
  fair, 397
CTL formula
  square brackets, 210
CTL*, 254

DAG, 70
dag, 364
dashed box
  flavour, 339
data structure, 123
de Morgan laws, 57, 216, 251
  for modalities, 313
deadlock, 178, 184, 215, 219
debugging systems, 222
debugging systems, 257
decision problem, 131
  of validity in predicate logic, 133
decision procedure, 55
declarative explanation, 26
declarative sentence, 2, 93
  truth value, 37
default case, 192
definition
  inductive, 33
description
  informal, 258
  language, 172, 174
Dijkstra, E., 283
directed graph, 136, 178, 364
  acyclic, 69, 364
  cycle, 364
disjunction, 4
  of literals, 55, 57
distributivity laws
  of box modality, 314
  of F connective, 185
  of propositional logic, 19, 60, 88
dividend, 302
domain assumption, 148
double negation-elimination, 352
double negation-introduction, 352

elimination rule, 6, 107
Emerson, E. A., 254
encoding, 128
entailment
  in program logics, 278
environment
  and non-determinism, 191
  for concurrent programs, 173
  for predicate logic formulas, 127
equality, 263
  intentional, 107
  program notation, 261
  structural, 153
  symbol, 107
equivalence relation, 321, 327, 339
equivalent formulas
  of basic modal logic, 314
  of CTL, 215–217

  of KT4, 327
  of KT45, 327
  of LTL, 184
  of predicate logic, 117
  of propositional logic, 16
  of relational mu-calculus, 410
exclusive-or, 382, 398
existential quantifier, 216
existential second-order logic, 139, 156
exists-elimination, 113
exists-introduction, 112

factorial
  of a natural number, 262
  program, 262, 284
fairness
  nested fixed points, 398
  symbolic model checking, 396
fairness constraint, 190, 197
  simple, 231, 252
FAIRNESS running, 204
Fibonacci numbers, 85
field index, 287
finite automata, 405
finite data structure, 222
first-order logic, 93
fixed point, 240
  greatest, 240, 241
  least, 240, 241
  semantics for CTL, 217, 238
flow of control, 261
Floyd, R., 269
for-statement, 299
forall-elimination, 109
forall-introduction, 110
formal
  path, 218
formula
  atomic, 175
  height, 44, 86
  Horn, 65
  ill-formed, 177
  immediate subformula, 223
  of basic modal logic, 314
  of CTL, 208
    atomic, 208
    ill-formed, 209
    well-formed, 209
  of LTL
    valid, 251
  of predicate logic, 100
  of propositional logic, 33, 50
    well-formed, 32, 33, 44
  of relational mu-calculus, 390
  positive, 328, 343, 348
  scheme, 312, 317
    **K**, 315
    in propositional logic, 312
    instance, 312
  subformula, 35
frame, 322
free for $x$ in $\phi$, 106, 109
Frege, G., 170
function
  in predicate logic, 124
  monotone, 240
    a non-example, 240
  nullary, 99
  recursive, 250
  SAT, 225, 227
    termination, 253
  SATaf, 228
  SATag, 253
  SATeg, 252
  SATeu, 229
  SATex, 228
  symbol, 96, 98, 157
    binary, 98
  translate, 250
function $\mathrm{pre}_\forall(X)$, 227
function $\mathrm{pre}_\exists(X)$, 227, 385
function $\mathrm{pre}_\forall(X)$, 385
function SAT
  correctness, 240
future
  excludes the present, 249, 353
  includes the present, 182, 249, 353
  whether it includes the present, 318

G-reachable, 338
  in $k$ steps, 338
Gödel, K., 96
Gentzen, G., 91
Global Assembly Cache, 149
grammar, 33
  clause, 269
guided simulation, 155

Halpern, J., 254
higher-order logic, 141
Hoare triple, 264
Hoare, C. A. R., 264, 269
Hodges, W., 170
Horn clause, 65, 139
hybrid rule, 343

if-statement, 280
implementation
  compliant, 143
implication, 4
  logical, 278
implies-elimination, 9
implies-introduction, 12
in-order representation, 35
inconsistency, 259
index, 132
induction
  course-of-values, 43
  hypothesis, 41, 42
  in model checking, 229
  mathematical, 40
inductive step, 41

infix notation, 125, 210
information
  negative, 343
input parameter, 61
integer
  expression, 260
integer label, 372
integer multiplication, 381
interface between logics, 277
interleaving
  formulas with code, 275
  transitions, 188, 194
introduction rules, 6, 107
introspection
  negative, 319, 326
  positive, 319, 326
intuitionistic logic, 30, 120, 327
invariants, 273
  discovering, 283
  SAT solver, 69
iterative squaring, 412

Jape, 170
justification, 276, 277, 329

Knaster-Tarski Theorem, 241
knowledge
  common, 333
  distributed, 335
  domain-specific, 102
  false, 321
  formula
    positive, 343
  idealised, 319, 327
  in a multi-agent system, 307
  modality, 335
  of agent, 307, 319
Kozen, D., 413
Kripke model, 167, 309
  as a counter example, 354
  for $KT45^n$, 336
Kripke, S., 309, 315

Löwenheim-Skolem Theorem, 138
label
  adding, 223
  deleting, 224
labelling
  AF, 223
  EG, 224
  $EG_C$, 231
  EU, 223
  EX, 223
labelling algorithm, 222
labelling function
  coding subsets, 383
  for Kripke model, 309
  for LTL model, 178
  frame does not have one, 322
language construct, 299
law of excluded middle, 25

LEM
  instance, 328
linear-time logic, 174
linear-time temporal logic, 175
Linear-time temporal logic (LTL), 175
literal, 55, 62
liveness, 190, 197
  property, 188, 189, 207, 230
logic engineering, 307, 316
logic programming, 49, 170
logical level, 278
logical variables
  of Hoare triple, 269
look-up table, 127, 140
  up-dated, 127
LTL, 175
LTLSPEC, 192, 204

machine state, 263
McMillan, K., 254
memoisation
  of computed OBDDs, 377
midcondition, 270, 276
minimal-sum section, 288
minimal-sum-section problem, 305
modal connective
  $C_G$, 335
  $K_i$, 335
modal logic, 306
  K, 326
  KT4, 327
  KT45, 326, 327
  normal, 326
  S4, 327
  S5, 326
modality, 308
  diamond, 308
  path, 220
model
  for propositional logic, 37
  of $KT45^n$, 336
  of basic modal logic, 309, 324
  of CTL, 310
    pictorial representation, 248, 249
  of intuitionistic propositional logic, 328
  of KT45, 337
  of $KT45^n$, 337
  of LTL
    pictorial representation, 179
  of predicate logic, 96, 124
  under-specified, 143
model checker, 174
model checking, 141, 173, 175, 256
  algorithm, 217, 225, 231, 318
  debugging, 394
  example, 182, 213
  with fairness constraints, 230
model-based verification, 172, 174
module, 265
modulo 8 counter, 408
*modus ponens*, 9

*modus tollens*, 10, 352
muddy-children puzzle, 342, 344
multiplicity constraint, 152
Mutex model
  pictorial representation, 188
mutual exclusion, 187

natural deduction
  extension to predicate logic, 95
  for modal logic, 332
  for temporal logic, 174
  inventor, 91
natural deduction rules
  for basic modal logic, 329
  for $KT45^n$, 340, 355
  for predicate logic, 107
  for propositional logic, 27
necessity
  logical, 308, 318
  physical, 318
negation, 4
negation-elimination (see
    'bottom-elimination'), 21
negation-introduction, 22
nested boolean quantification, 394
network
  synchronous, 174
no strict sequencing, 188, 189, 215
node
  initial, 364
  leaf, 103
  non-terminal, 361
  shared, 70
  terminal, 362, 364
non-blocking protocol, 188, 189, 215
non-determinism, 190
non-termination, 262
normal form, 53, 55
  conjunctive, 55, 360
  disjunctive, 360
  negation, 60
    CTL*, 250
    LTL, 186, 246
  product-of-sums, 406
  sum-of-products, 404
not-elimination, 21
not-introduction, 22

OBDD, 367
  absence of redundant variables,
    370
  canonical form, 369
  complementation, 385
  definition, 367
  extensions, 381
  for $\mathrm{pre}_{\exists}(X)$, 387
  for $\mathrm{pre}_{\forall}(X)$, 387
  integer multiplication, 381
  intersection, 385
  limitations, 381
  memoisation, 377
  nested boolean quantification, 380
  of a transition relation, 386
  of an even parity function, 370
  of the odd parity function, 400
  optimal ordering, 406
  reduced, 368
    unique representation, 368
  reduced one for logical 'iff', 400
  representing subsets, 383
  running time of algorithms
    upper bounds, 380
  sensitivity of size, 370
  synthesis of boolean formula, 380
  test
    for implication, 372
    for satisfiability, 372
    for semantic equivalence, 370
    for validity, 372
  union, 385
  variations, 381
odd parity function, 400
omniscience
  logical, 319
or-elimination, 17
or-introduction, 16
overloading
  of $\models$, 129
  of proof rules, 107

parity function
  even, 369
    as OBDD, 370
parity OBDD, 381
parse tree
  for a predicate logic formula, 103
  of a term, 159
  of a basic modal logic formula, 307
  of a CTL formula, 210
  of propositional logic formula, 34
  root, 35
  subtree, 35
  underspecified, 312
partial correctness, 265
partial order reduction, 229
pattern
  `checkEU` $(f, g)$, 398
  `checkEX` $(f)$, 397
pattern matching, 6, 111, 279
place holder, 94
possibility, 316
  logical, 308
possible world
  semantics, 315
Post correspondence problem, 132
postcondition, 151
  in program logic, 264
Prawitz, D., 91
precondition, 151
  in program logic, 264
    weakest, 276
  of algorithm, 63

predicate, 93
  binary, 95
  number of arguments, 95
  unary, 95
predicate logic, 93
  consistent set of formulas, 129
  satisfiability, 129
  semantic entailment, 129
  validity, 129
prefix, 126
  notation, 210
  ordering, 126
premise, 270
Prior, A., 254
problem
  instance, 132
  reduction, 132
procedural interpretation, 26
process
  concurrent, 187
  instantiation, 206
processor, 257
program
  behaviour, 264
  bug, 257
  code, 276
  construct, 261
  correctness, 63, 67, 225
  derived, 299
  diverging, 266
  documentation, 257
  environment, 258
  finite-state, 358
  fragment, 262
  logic, 275
  methodology, 259
  procedures, 263
  sequential, 256
  termination, 67, 89, 242, 265
  variable, 227, 268
  verification, 270
    formal, 260
program execution, 316, 319
programming by contract, 296
  Eiffel, 296
programming language
  imperative, 259
proof
  box
    for →i, 11
    for forall-introduction, 110
    for modal logic, 329
    opening, 28
    side by side, 22
  by contradiction, 24
  calculus, 256, 260
  construction, 269
  constructive, 120
  dashed box, 329, 340
  fragment, 278
  indirect, 29
  of correctness, 239
  of termination, 266
  partial, 281
  partial correctness, 269, 300
  search, 49
  solid box, 329
  strategy, 115, 265
  subproof, 272
  tableaux, 269
  theory, 93, 122, 174
  total correctness, 292
proof rules, 5
  for implication, 273
  for assignment, 269
  for conjunction, 6
  for disjunction, 16
  for double negation, 8
  for equality, 108
  for existential quantification, 112
  for if-statements, 272, 280
    modified, 281
  for implication, 12, 277
  for $KT45^n$, 339
  for negation, 20
  for quantifiers, 112
  for sequential composition, 269, 275
  for universal quantification, 109
  for while-statements, 272, 282, 287
  schema, 111
  subformula property, 113
proof tableaux
  complete, 291
proof-based verification, 172, 256
proposition, 2
propositional logic, 93
protocol, 187, 188
provability
  undecidability of predicate logic, 136

quantifier, 310, 313
  equivalences, 185
  in predicate logic, 94
    binding priorities, 101
    equivalences, 130
    meaning, 123
Quielle, J., 254

reachability, 136, 137
reasoning
  about knowledge, 326, 331
  constructive, 29
  in an arbitrary accessible world, 329
  informal, 343
  quantitative, 259
  unsound, 280
record
  field, 193
recursion
  mutual, 218
recursive call, 280
*reductio ad absurdum*, 24, 119
reduction to absurdity, 24
reflexive, transitive closure, 167

regular language, 405
relation
  binary, 178
  Euclidean, 321, 327
  functional, 321
  linear, 321
  reflexive, 140, 320, 324
    as formula, 109
  serial, 320, 353
  symmetric, 320
    as formula, 109
  total, 321
  transition, 178
  transitive, 140, 320, 324
    as formula, 109
relational mu-calculus
  fixed-point operators, 392
requirement
  informal, 258, 263, 288
requirements, 142
restriction, 374
right-associative, 5
root of a parse tree, 135
rule
  derived, 23
  hybrid, 10
Russell's paradox, 165

safety property, 187, 189, 207
SAT solver
  cubic, 76
  forcing rules, 71
  permanent marks, 75
  temporary marks, 74
satisfaction
  in a frame, 322
  in a frame for $KT45^n$, 337
satisfaction relation
  for relational mu-calculus, 391
  for basic modal logic, 310
  for KT45, 337
  for LTL, 180
  for partial correctness, 265
  for predicate logic, 128
  for relational mu-calculus, 391
  for total correctness, 266
satisfiability, 360
  3SAT, 406
  deciding, 65
  of a propositional logic formula, 85
  undecidability of predicate logic, 135
SCC
  fair, 232
scheduler
  fair, 197
scope
  of a dummy variable, 117
  of a variable, 103, 113
  of an assumption, 28, 113, 329
search space, 113, 133
second-order logic, 141
semantic entailment
  for predicate logic, 141
  for basic modal logic, 313
  for KT4, 328
  for normal modal logics, 326
  for predicate logic, 96
  for propositional logic, 46
  for relational mu-calculus, 410
semantic equivalence, 39
semantics
  of $\mu Z.f$, 392
  of $\nu Z.f$, 393
  of basic modal logic, 310
  of boolean quantification, 392
  of CTL, 211
  of EG, 239
  of equality, 131
  of predicate logic, 122
  of propositional logic, 38
  of relational mu-calculus, 391
  of Until, 181
sentence
  atomic, 4
  components, 93
  declarative, 93
  in predicate logic, 128
sequent, 5
  invalid, 116
Shannon expansion, 374
side condition, 108, 110
Sifakis, J., 254
small scope hypothesis, 143
SMV, 254
  main program for ABP, 207
  module, 193
    `receiver`, 205
    `sender`, 204
    for channel, 206
    instantiation, 193
  process, 389
  program
    example, 192
    for Mutex, 195
  specification, 192
software
  life-cycle, 142
  micromodel, 142
  reliability, 149
  requirements, 142
  specification, 142
  validation, 142
soundness
  of forall-elimination, 109
  of natural deduction
    basic modal logic, 354
    predicate logic, 96, 122
    propositional logic, 45
  of program logics, 267
  of proof rule for while-statements, 282
  of the substitution principle, 108

specification
  for ABP, 207
  formal, 259
  informal, 259
  language, 172
  of a predicate, 157
  patterns, 254
  practical pattern, 183, 215
  truth table, 58
specifications, 191
Spin, 254
state
  critical, 188
  explosion, 229
  explosion problem, 254
  fair, 397
  final, 142
  formula, 218
  global, 188
  graph, 180
  initial, 142, 189, 222, 247, 252, 264
  non-critical, 188
  of a system, 269
  of core program, 264
  reachable, 247
  resulting, 263, 299
  space, 229
  splitting states, 190
  transition, 142
  trying, 188
state machine, 142
storage
  location, 288
  state, 261
store
  of core program, 264
string, 247, 307
  binary, 126, 132
  empty, 126
strongly connected component, 225
structural equality, 153
structural induction, 44, 51
subformula, 178
substitution
  in predicate logic, 105
  instance, 323
  instance of tautology, 314
  principle, 108
symbolic model checking, 383
syntactic
  domain, 260, 261
syntax
  of basic modal logic, 307
  of boolean expressions, 261
  of boolean formulas, 398
  of CTL, 208
  of CTL*, 218
  of KT45$^n$, 335
  of LTL, 175
  of predicate logic, 100
  of propositional logic, 33
  of relational mu-calculus, 390
  of terms, 99
system
  asynchronous, 254
    interleaving model, 389
    simultaneous model, 389
  axiomatic, 91
  commercially critical, 172, 257
  component, 206
  concurrent, 173
  debugging, 174
  description, 193
  design, 174
  development, 173
  elevator, 184, 215
  finite-state, 256
  hybrid, 277
  infinite-state, 256
  mission-critical, 172
  multi-agent, 331
  physical, 175
  reactive, 173, 257, 358
  safety-critical, 172, 257
  transition, 174
  verification, 256

tautology, 50
temporal connective
  AF, 212
  AG, 211
  AU, 212
  AX, 211
  EF, 212
  EG, 211
  EU, 212
  EX, 211
temporal connectives, 176
temporal logic, 174, 306
term, 99
  interpretation, 128
term-rewriting system, 170
termination
  Collatz $3n + 1$, 295
  proof, 266
*tertium non datur*, 25
theorem, 13
  prover, 106, 136
  proving, 170
time
  continuous, 174
  discrete, 174
top
  marking, 66
total correctness, 265, 266
transition relation, 178
  for SMV programs, 388
transition system, 174
  of ABP program, 247
  of Mutex code, 198
  of SMV program, 192
  unwinding, 180, 212, 222

translation
  English into predicate logic, 95, 101
tree
  infinite, 180, 212
truth
  dynamic, 174
  mode, 306, 308
  of knowledge, 326
  static, 174
  value
    for predicate logic, 127
    for propositional logic, 3
truth table
  for conjunction, 37
truth tables, 38
type, 12, 327
  checking, 12
  theory, 170

unary connective, 307
undecidability
  of provability, 136
  of satisfiability, 135
  of validity in predicate logic, 133
universal quantification, 268
universal quantifier, 216
universal second-order logic, 140, 156
universe of concrete values, 124
unreachability, 140
unsound sequent, 164
Until
  in natural language, 182
  negating, 187
updated valuation, 391

valid sequent
  of modal logic, 330
  partial correctness, 267
  total correctness, 267
validity
  in basic modal logic, 314
  in KT45$^n$, 339
  in propositional logic, 85
  undecidability in predicate logic, 133
valuation
  for propositional logic, 37
  in predicate logic, 123
  in relational mu-calculus, 391
value
  initial, 206, 268, 269
Vardi, M., 254
variable, 94, 260
  boolean, 229, 247, 358
  bound, 103
  capture, 106
  dummy, 110
  free, 103
  local, 263
  logical, 268, 290
variable ordering
  compatible, 368
  list, 367
variant, 293
verification
  full, 173
  method, 172
  of communication protocols, 175
  of hardware, 175
  of software, 175
  of systems, 256
  post-development, 173, 257
  pre-development, 173, 257
  process, 271
  program, 270
  property, 173
  property-oriented, 256
  semi-automatic, 256
  techniques, 172

weakest precondition, 276
while-statement, 261, 262
  body, 273, 282, 286
  non-termination, 292
wise-men puzzle, 342
Wolper, P., 254
word
  empty, 126
world
  accessible, 309
  possible, 309, 336

year-2000 problem, 258