The formula φ, however, is not an inductive invariant. The state $(1, 0, 0)$ satisfies the formula φ, and has a transition to the state $(2, 0, 1)$, which *does not* satisfy the formula φ.

Consider the formula ψ given by $(z = 0 \ \wedge \ x = y) \vee (z = 1 \ \wedge \ x = y + 1)$. Observe that if a state s satisfies ψ, it must satisfy one of the disjuncts in ψ, and thus, must satisfy either $(x = y)$ or $(x = y + 1)$, and thus, must satisfy φ. Thus, the property ψ is stronger than φ. The initial state $(0, 0, 0)$ satisfies ψ. Consider a state s that satisfies ψ. Then s satisfies either $(z = 0 \ \wedge \ x = y)$ or $(z = 1 \ \wedge \ x = y + 1)$. In the former case, executing a transition from the state s increments x and sets z to 1, and thus, the resulting state satisfies $(z = 1 \ \wedge \ x = y + 1)$. By a similar reasoning if the state s satisfies $(z = 1 \wedge x = y + 1)$, then executing one transition from it leads to a state that satisfies $(z = 0 \wedge x = y)$. It follows that if there is a transition from the state s to state t, then the state t must satisfy ψ. Thus, the property ψ is an inductive invariant. ■

Solution 3.7: The transition system $\texttt{Mult}(m, n)$ has a transition from state $s = (\texttt{loop}, 0, k)$ to state $t = (\texttt{stop}, 0, k)$, for every natural number k. If $k \neq m \cdot n$, then the state s satisfies the property φ given by $(mode = \texttt{stop}) \rightarrow (y = m \cdot n)$, but the state t does not. It follows that the property φ is not an inductive invariant.

Consider the property ψ given by

$$[(mode = \texttt{loop}) \ \wedge \ y = (m - x) \cdot n] \ \vee \ [(mode = \texttt{stop}) \ \wedge \ (y = m \cdot n)].$$

If a state s satisfies the formula ψ, and the value of $mode$ in state s is $\texttt{stop}$, then the state s must satisfy $(y = m \cdot n)$. It follows that a state satisfying the property ψ must satisfy the property φ.

The initial state $(\texttt{loop}, m, 0)$ satisfies the formula ψ. Now consider a state s satisfying ψ. Suppose the value of $mode$ in state s is $\texttt{loop}$. We know that the condition $s(y) = (m - s(x)) \cdot n$ holds. If $s(x) > 0$ then executing a transition in state s leaves the mode unchanged, decrements x, and increases y by n. That is, $t(mode) = \texttt{loop}$, $t(x) = s(x) - 1$, and $t(y) = s(y) + n$. It is easy to establish that $t(y) = (m - t(x)) \cdot n$ also holds, and thus the state t satisfies ψ. If $s(x) = 0$, then the condition $s(y) = m \cdot n$ holds, and executing a transition in state s updates the mode to $\texttt{stop}$ and leaves the variables x and y unchanged. In this case also, the resulting state t satisfies the property ψ. If the value of $mode$ in state s is $\texttt{stop}$, then there is no transition out of state s. It follows that the property ψ is preserved by transitions of the system $\texttt{Mult}(m, n)$, and it is an inductive invariant. ■

Solution 3.8: The state $(\texttt{on}, 0)$ satisfies the property φ, and has a transition to the state $(\texttt{off}, -1)$ which does not satisfy the property φ. This shows that the property is not an inductive invariant.

Consider the property ψ given by

$$(mode = \texttt{off} \ \wedge \ x \geq 0) \ \vee \ (mode = \texttt{on} \ \wedge \ x > 0).$$

We will first show that the property ψ is stronger than the property φ. Consider a state s that satisfies ψ. Depending of the value of $mode$ in state s, either $x \geq 0$ or $x > 0$ holds in the state s. In either case the condition φ is satisfied.

The initial state $(\texttt{off}, 0)$ satisfies the property ψ. Now consider a state s that satisfies ψ. To prove that the property ψ is an inductive invariant, we need to establish that, if there is a transition from the state s to state t, then the state t also satisfies ψ. Suppose $s = (\texttt{off}, a)$. Then, it must be the case that $a \geq 0$. The state s has 2 successor states $t_1 = (\texttt{off}, a+1)$ and $t_2 = (\texttt{on}, a+1)$. Clearly, the state t_1 satisfies $(mode = \texttt{off} \ \wedge \ x \geq 0)$, and the state t_2 satisfies $(mode = \texttt{on} \wedge x > 0)$. Thus, both the states satisfy ψ. Now suppose $s = (\texttt{on}, a)$. Then, it must be the case that $a > 0$. The state s has only one outgoing transition to the state $t = (\texttt{off}, a-1)$, and this state satisfies $(mode = \texttt{off} \ \wedge \ x \geq 0)$ (since $a - 1 \geq 0$), and thus, the property ψ. ■

Solution 3.9 : 1. The property is an invariant, and in fact, and an inductive invariant. In the initial state $near_E$ equals 0 and $mode_E$ is `away`. Consider an arbitrary state where $near_E$ equals 0 and $mode_E$ is `away`. The value of $mode_E$ changes (to `wait`) exactly when the condition (out_E ? `arrive`) holds, but this coincides with the condition under which the controller changes $near_E$ to a non-zero value, and thus, in the resulting state both the conditions $(near_E = 0)$ and $(mode_E = \texttt{away})$ are false, and the equivalence continues to hold. Now consider an arbitrary state where $near_E$ equals 1 and $mode_E$ is not `away`. During a transition the condition $(mode_E = \texttt{away})$ can become true only when (out_E ? `leave`) holds, which is precisely the condition under which the controller changes $near_E$ to 0.

2. The property is an invariant, but is not an inductive invariant. Consider a state in which $mode_E$ equals `bridge`, $east$ equals `green`, and $near_E$ equals 0 (such a state is actually unreachable). This state satisfies the property, This state has a transition to a state in which $mode_E$ stays unchanged, $near_E$ stays unchanged, but $east$ gets updated to `red`, violating the property.

3. The property is an inductive invariant. The initial state satisfies this property. By examining the update-code in figure 3.8 observe that the variable $east$ is updated to `green` only under the condition $(west = \texttt{red})$, and the variable $west$ is updated to `green` only under the condition $(east = \texttt{red})$. It follows that executing this code in a state where at least one of $east$ or $west$ equals `red` cannot lead to a state with both variables equal to `green`. ■

Solution 3.10 : The reactive component `Switch` has 12 reachable states: $(\texttt{off}, 0)$, and $(\texttt{on}, n)$, for $0 \leq n \leq 10$. The state $(\texttt{off}, 0)$ has two transitions, to itself and the result will not reflect the transitions of the composed component correctly. As a concrete example, suppose the component C_1 has a state variable x of type `nat`, and an output variable y of type `bool`. In each transition, either the output is 0 and x stays unchanged, or the output is 1 and x is incremented by 1. Thus, the reaction formula φ^1_R is $(x' = x \wedge y = 0) \vee (x' = x+1 \wedge y = 1)$, and the transition formula φ^1_T is $(x' = x) \vee (x' = x+1)$. The component C_2 has a state variable z of type `nat`, and the input variable y. If the input y is 0, z stays unchanged, and if the input y is 1, z is incremented by 1. Thus, the reaction formula φ^2_R is $(z' = z \wedge y = 0) \vee (z' = z+1 \wedge y = 1)$, and the transition formula φ^2_T is $(z' = z) \vee (z' = z+1)$. When we compose C_1 and C_2, in each round either y is 0 and both x and z stay unchanged, or y is 1 and both x and z are incremented. However, the conjunction $\varphi^1_T \wedge \varphi^2_T$ is the formula

$$[(x' = x) \vee (x' = x+1)] \wedge [(z' = z) \vee (z' = z+1)].$$

This does not capture the transitions of $C_1 \| C_2$ accurately as it allows the possibility of x staying unchanged while z gets incremented. ■

Solution 3.16: Conjunction of the given region A and the transition formula gives

$$(x' = x+1) \wedge (y' = x) \wedge (0 \leq x \leq 4) \wedge (y \leq 7).$$

The existential quantification of the unprimed variables leads to $(x' = y'+1) \wedge (0 \leq y' \leq 4)$. Renaming the primed variables to their unprimed counterparts gives the desired post-image: $(x = y+1) \wedge (0 \leq y \leq 4)$. ■

Solution 3.17: The transition formula is given by:

$$[(x < y) \wedge (x' = x+y) \wedge (y' = y)] \vee [(x \geq y) \wedge (x' = x) \wedge (y' = y+1)].$$

To obtain the post-image of the region $(0 \leq x \leq 5)$, we first conjoin this formula with the transition formula. Existentially quantifying y gives

$$[(x < y') \wedge (x' = x+y') \wedge (0 \leq x \leq 5)] \vee [(x \geq y'-1) \wedge (x' = x) \wedge (0 \leq x \leq 5)].$$

Existentially quantifying x from this formula gives

$$[(x' < 2y') \wedge (0 \leq x' - y' \leq 5)] \vee [(x' \geq y'-1) \wedge (0 \leq x' \leq 5)].$$

Renaming the primed variables to the corresponding unprimed ones gives the desired result:

$$[(x < 2y) \wedge (0 \leq x - y \leq 5)] \vee [(x \geq y-1) \wedge (0 \leq x \leq 5)].$$

■

Solution 3.18: Given a region A, to compute its pre-image, we first rename the unprimed variables to primed variables, and then intersect it with the transition region *Trans* over $S \cup S'$ to obtain all the transitions that lead to the states in

A. Then, we project the result onto the set S of unprimed state variables by existentially quantifying the variables in S'. Thus the pre-image operator `Pre` is defined as:

$$\texttt{Pre}(A, \mathit{Trans}) \;=\; \texttt{Exists}(\texttt{Conj}(\texttt{Rename}(A, S, S'), \mathit{Trans}), S').$$

The backward-search algorithm is symmetric to the algorithm in Figure 3.18. The region *Reach* contains all the states from which a state satisfying the property φ has been discovered to be reachable. It initially contains the states that satisfy φ, and in each iteration, states from which there is a transition to a state already in *Reach*, are added using the pre-image computation. At any step, if the region *Reach* contains an initial state, the algorithm has discovered an execution from an initial state to a state satisfying φ, and can terminate.

Input: A transition system T given by a region *Init* for initial states and a region *Trans* for transitions, and a property φ.
Output: If φ is reachable in T, return 1, else return 0.

```
reg Reach := φ;
reg New := φ;
while IsEmpty(New) = 0 do {
  if IsEmpty(Conj(New, Init)) = 0 then return 1;
  New := Diff(Pre(New, Trans), Reach);
  Reach := Disj(Reach, New);
  };
return 0.
```

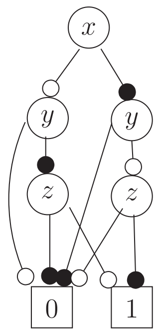
■

Solution 3.19 : During the execution of the algorithm of figure 3.18, let New_1 be the value of the region *New* at the beginning of the first iteration of the while-loop, let New_2 be its value at the beginning of the second iteration of the loop, and so on. Suppose during the ith iteration of the while-loop, the algorithm discovers a state satisfying the property φ, that is, the intersection of the regions New_i and φ is non-empty. To return a witness execution, we first choose a specific state, say, s_i that belongs to both New_i and φ. Then, we need to find a state that belongs to the region New_{i-1} and has a transition to state s_i. This can be achieved by computing the set of predecessors of the state s_i, intersect this set with the region New_{i-1}, and select a state s_{i-1} in this intersection (note that this intersection is guaranteed to be a non-empty region since the region New_i was obtained by applying the post-image operation to the region New_{i-1}). We can repeat this process till an initial state in the region New_1 is chosen. The modified algorithm is shown below. The sequence of regions $New_1, New_2, \ldots$ is stored using the stack *Frontiers*. The algorithm uses one new operation on regions: given a non-empty region A, `SelectState`(A) returns one state belonging to A (the specific choice does not matter). The pre-image computation operation `Pre` is the same as the one described in exercise 3.18, except it is now applied to a region containing a single state.

```
reg Reach := Init;
reg New := Init;
stack(reg) Frontiers := EmptyStack;
while IsEmpty(New) = 0 do {
  if IsEmpty(Conj(New, φ)) = 0 then {
    stack(state) Exec := EmptyStack;
    state s := SelectState(Conj(New, φ));
    Push(s, Exec);
    while IsEmpty(Frontiers) = 0 {
      s := SelectState(Conj(Pop(Frontiers), Pre(s, Trans)));
      Push(s, Exec);
      };
    return Exec
    };
  Push(New, Frontiers);
  New := Diff(Post(New, Trans), Reach);
  Reach := Disj(Reach, New);
  };
return 0.
```

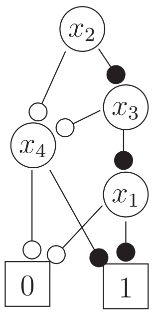
■

Solution 3.20 : The ROBDD is shown below:



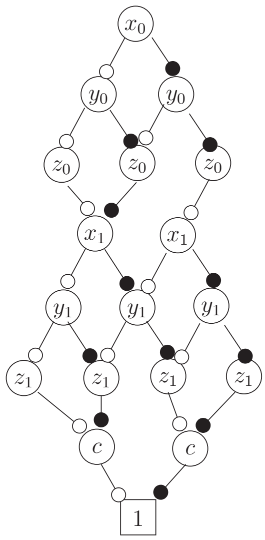
■

Solution 3.21 : The ROBDD for the variable ordering $x_2 < x_3 < x_4 < x_1$ is shown below:



For each variable x_i, there is only one vertex labeled with x_i. Given that the Boolean function captured by the formula does depend on all the four variables, the ROBDD could not possibly have fewer than 4 internal vertices no matter which variable ordering we pick. Thus, this is the smallest possible ROBDD. ■

Solution 3.22: The natural variable ordering is $x_0 < y_0 < z_0 < x_1 < y_1 < z_1 < c$. The corresponding ROBDD is shown below:



Note that to simplify the drawing, we have omitted the terminal node 0, and the edges that lead to it (for example, the right-edge of the left node labeled with c and the left-edge of the right node labeled with c). ■

Solution 3.23: The algorithm for existential quantification uses the algorithm for computing disjunction of ROBDDs: given two ROBDDs B and B', the routine $Disj(B, B')$ returns the ROBDD representation of the function $f(B) \vee f(B')$. The implementation of $Disj(B, B')$ follows the same outline as the algorithm for $Conj(B, B')$ in figure 3.26.