Possible Solutions for Exercise Sheet 3

Exercise 1.

Let $F = (x \vee y) \to z$ and $G = x \leftrightarrow (y \wedge z)$, where $x < y < z$.

(a) Draw the BDDs of F and G.

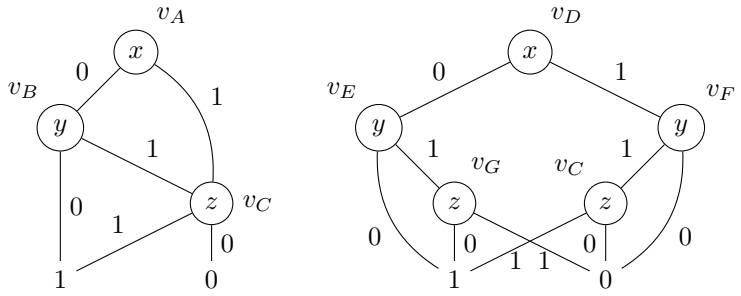
(b) During the lecture an algorithm implementing the OR-operation on BDDs has been presented. Use this algorithm to construct a BDD for $F \vee G$.

(c) Modify this algorithm to receive the BDD for $F \wedge G$.

(d) Let $H = ((z \leftrightarrow (x \oplus y)) \wedge (w \leftrightarrow (y \vee z)) \vee x \vee (u \leftrightarrow (x \vee w))$. Draw the BDD of H. How many satisfiable assignments exist for H? How could one read this directly from the BDD?
Hint: For each node give the number of satisfying assignments for the subtree starting at that node. Start with the "lowest" one.
Hint #2: You may either calculate the BDD yourself or may use one of the tools presented on the homepage.

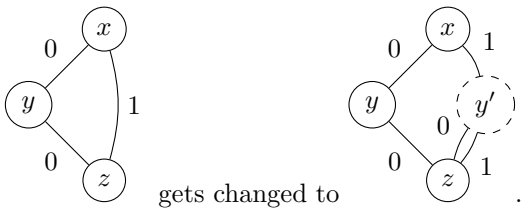
Solution:

(a) The BDDs for F (left) and G (right) are: The nodes are assigned names v_x for some x, which are used in the coming exercise. Note that v_C has been assigned twice as it indeed represents the same node (in a multiBDD, it would only occur once).



The nodes are assigned names v_x for some x, which are used in task b). Note that v_C has been assigned twice as it indeed represents the same node (if F and G would be in one multiBDD, it would only occur once).

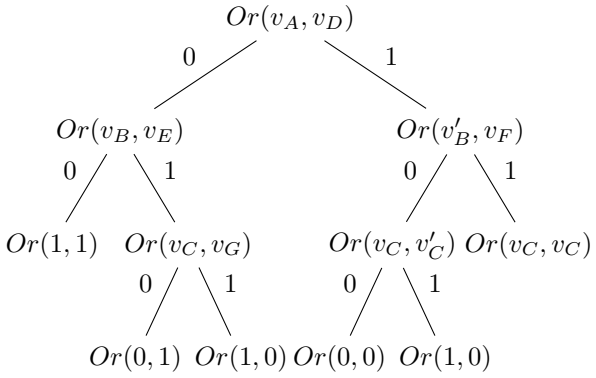
(b) Model the call-stack of the recursive *Or*-algorithm. If an edge skips a/several level/s, add virtual nodes for each level on this edge, where both outgoing edges go to the following node. So for example



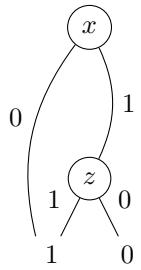
.

This ensures, that during the application of the algorithm you will always work with the same variable on both nodes.

The call-stack (with v'_B and v'_C being virtual nodes):



The final BDD for $F \vee G$ then is:

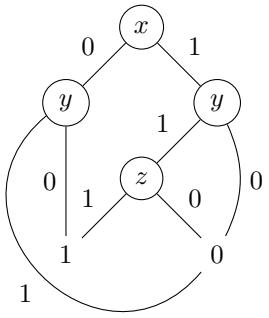


(c) The only things, which need a change are the special cases at the beginning of the algorithm:

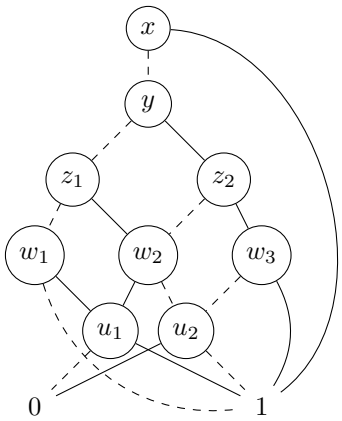
if $v_F = K_0$ **or** $v_G = K_0$ **then return** K_0
else if $v_F = v_G = K_1$ **then return** K_1

The rest of the algorithm remains unchanged (except, of course, the name of the recursive calls).

The call-stack then is the same as for $F \vee G$, and the final BDD is:



(d) The BDD might come in different forms as the order of variables is not specified. One such a BDD might be (dotted lines are negative edges, full strokes are positive edges):



To calculate the number of satisfying assignments, we assign each node a *weight*, which states the number of satisfying assignments in this subgraph. The definition follows a set of three rules:

- The node 1 receives the weight 1, the node 0 receives the weight 0.

- Each other node receives the sum of the weight of the nodes of each outgoing edge as the weight.
- If an edge skips a level of variables, the weight gets doubled for each layer. If you introduce virtual nodes on this edge for each level with two outgoing edges to the very same node, you see why this is the case.

Hence the weight for the nodes of the BDD above is (from bottom to top):

1	1	(by def.)
0	0	(by def.)
u_1	1	$(0 + 1)$
u_2	1	$(0 + 1)$
w_1	3	$(u_1 + 2 * 1)$
w_2	2	$(u_1 + u_2)$
w_3	3	$(u_2 + 2 * 1)$
z_1	5	$(w_1 + w_2)$
z_2	5	$(w_2 + w_3)$
y	10	$(z_1 + z_2)$
x	26	$(y + 16 * 1)$

The weight of x then is the total number of satisfiable assignments.

Exercise 2.

Let us fix a directed tree $T = (V, <)$ with root r such that moreover V is countable. Obviously, if there is an infinite path from r, then this infinite path witness that V is infinite.

Conversely, let us assume that V is infinite. As suggested in the exercise sheet, let us introduce the propositional variable x_v for each $v \in V$. Moreover, let $V_i = \{v \in V \mid r <^i v\}$ denote the nodes with distance precisely i from the root r, for each $i \geq 0$. One easily proves by induction on i that each V_i is finite. For each $i \geq 0$ and for each $v \in V_i$ such that $r = u_0 < u_1 < \cdots < u_i = v$ let us define the formula F_v that, among the variables $\{x_w \mid w \in V_j, j \in \{0, \ldots, i\}\}$, "selects" precisely the variables corresponding to the unique path from r to v:

$$F_v \;=\; \bigwedge_{j \in \{0,\ldots,i\}} \left(x_{u_j} \wedge \bigwedge_{w \in V_j \setminus \{u_j\}} \neg x_w \right)$$

For each "level" $i \geq 0$, let the formula G_i express that the path to precisely one node in V_i is "selected":

$$G_i \;=\; \bigvee_{v \in V_i} F_v$$

The formulas F_v and G_i are well-defined since each V_i is finite. Finally let us choose the set of formulas Γ as follows:

$$\Gamma \;=\; \{G_i \mid i \geq 0\}$$

Note that each finite subset of Γ is satisfiable. Why?

By the compactness theorem Γ itself is satisfiable. Let $\mathcal{A}$ be a model of Γ, thus $\mathcal{A} \models G_i$ for each $i \geq 0$. Clearly, $\mathcal{A}$ "selects" precisely the variables corresponding to some infinite path starting from r.

Exercise 5.

We will show that for each suitable structure $\mathcal{A}$, for each $u \in U^{\mathcal{A}}$, for each formula G and for each variable y that does not occur in G

$$\mathcal{A}_{[x/u]} \models G \quad \Longleftrightarrow \quad A_{[y/u]} \models G[x/y]. \tag{1}$$

Proving (1) is sufficient for the statement of the exercise when $Q = \forall$ (analogously for $Q = \exists$) since it implies the following for each suitable structure $\mathcal{A}$:

$$\begin{aligned}
\mathcal{A}_{[x/u]} \models \forall x G \quad &\Longleftrightarrow \quad \text{for all } v \in U^{\mathcal{A}_{[x/u]}}\colon \mathcal{A}_{[x/u][x/v]} \models G \\
&\Longleftrightarrow \quad \text{for all } v \in U^{\mathcal{A}_{[x/u]}}\colon \mathcal{A}_{[x/v]} \models G \\
&\overset{(1)}{\Longleftrightarrow} \quad \text{for all } v \in U^{\mathcal{A}}\colon \mathcal{A}_{[y/v]} \models G[x/y] \\
&\Longleftrightarrow \quad \text{for all } v \in U^{\mathcal{A}_{[y/u]}}\colon \mathcal{A}_{[y/u][y/v]} \models G[x/y] \\
&\Longleftrightarrow \quad \mathcal{A}_{[y/u]} \models \forall y G[x/y]
\end{aligned}$$

Let us prove (1) by structural induction on G.

Induction base. Assume $G = R(t_1, \ldots, t_n)$, where R is an n-ary relational symbol and $t_1, \ldots t_n$ are terms. To deal with this case we prove that for each term t in which y does not occur we have

$$\mathcal{A}_{[x/u]} \;=\; \mathcal{A}_{[y/u]}(t[x/y]) \tag{2}$$

by induction on the structure of t.

- *Induction base.* We make a case distinction.
  - Case 1: $t = a$ for some constant symbol a. Then $\mathcal{A}_{[x/u]}(a) = \mathcal{A}(a) = \mathcal{A}_{[y/u]}(a[x/y])$.
  - Case 2: $t = z$ for some variable $z \neq x$. Then $\mathcal{A}_{[x/u]}(z) = \mathcal{A}(z) = \mathcal{A}_{[y/u]}(z[x/y])$.
  - Case 3: $t = x$. Then $\mathcal{A}_{[x/u]}(x) = u = \mathcal{A}_{[y/u]}(y) = \mathcal{A}_{[y/u]}(x[x/y])$.
- *Induction step.* Assume $t = (r_1, \ldots, r_k)$ for some k-ary function symbol f and some terms $r_1, \ldots, r_k$. Then due to $f^{\mathcal{A}_{[x/u]}} = f^{\mathcal{A}_{[y/u]}}$ we have

$$\begin{array}{rcl}
\mathcal{A}_{[x/u]}(t) & = & f^{\mathcal{A}_{[x/u]}}(\mathcal{A}_{[x/u]}(r_1), \ldots, \mathcal{A}_{[x/u]}(r_k)) \\
& \overset{\text{IH}}{=} & f^{\mathcal{A}_{[x/u]}}(\mathcal{A}_{[y/u]}(r_1[x/y]), \ldots, \mathcal{A}_{[y/u]}(r_k[x/y])) \\
& = & f^{\mathcal{A}_{[y/u]}}(\mathcal{A}_{[y/u]}(r_1[x/y]), \ldots, \mathcal{A}_{[y/u]}(r_k[x/y])) \\
& = & \mathcal{A}_{[y/u]}(t[x/y])
\end{array}$$

Let us now jump back to the outer induction hypothesis. Note that $R^{\mathcal{A}_{[x/u]}} = R^{\mathcal{A}_{[y/u]}}$. Hence

$$\begin{array}{rcl}
\mathcal{A}_{[x/u]} \models R(t_1, \ldots, t_n) & \iff & (\mathcal{A}_{[x/u]}(t_1), \ldots, \mathcal{A}_{[x/u]}(t_n)) \in R^{\mathcal{A}_{[x/u]}} \\
& \iff & (\mathcal{A}_{[x/u]}(t_1), \ldots, \mathcal{A}_{[x/u]}(t_n)) \in R^{\mathcal{A}_{[y/u]}} \\
& \overset{(2)}{\iff} & (\mathcal{A}_{[y/u]}(t_1[x/y]), \ldots, \mathcal{A}_{[y/u]}(t_n[x/y])) \in R^{\mathcal{A}_{[y/u]}} \\
& \iff & \mathcal{A}_{[y/u]} \models R(t_1, \ldots, t_n)[x/y]
\end{array}$$

Induction step.

- $G = G_1 \wedge G_2$. Then

$$\begin{array}{rcl}
\mathcal{A}_{[x/u]} \models G & \iff & \mathcal{A}_{[x/u]} \models G_1 \text{ and } \mathcal{A}_{[x/u]} \models G_2 \\
& \overset{\text{IH}}{\iff} & \mathcal{A}_{[y/u]} \models G_1[x/y] \text{ and } \mathcal{A}_{[y/u]} \models G_2[x/y] \\
& \iff & \mathcal{A}_{[y/u]} \models G[x/y]
\end{array}$$

- $G = G_1 \vee G_2$. Then

$$\begin{array}{rcl}
\mathcal{A}_{[x/u]} \models G & \iff & \mathcal{A}_{[x/u]} \models G_1 \text{ or } \mathcal{A}_{[x/u]} \models G_2 \\
& \overset{\text{IH}}{\iff} & \mathcal{A}_{[y/u]} \models G_1[x/y] \text{ or } \mathcal{A}_{[y/u]} \models G_2[x/y] \\
& \iff & \mathcal{A}_{[y/u]} \models G[x/y]
\end{array}$$

- $G = \neg G'$. Then

$$\begin{aligned} \mathcal{A}_{[x/u]} \models G &\iff \mathcal{A}_{[x/u]} \not\models G' \\ &\overset{\text{IH}}{\iff} \mathcal{A}_{[y/u]} \not\models G'[x/y] \\ &\iff \mathcal{A}_{[y/u]} \models G[x/y] \end{aligned}$$

- $G = \exists z G'$, where $z \neq x$. Then

$$\begin{aligned} \mathcal{A}_{[x/u]} \models G &\iff \text{there exists some } v \in U^{\mathcal{A}_{[x/u]}} \text{ such that } \mathcal{A}_{[x/u][z/v]} \models G' \\ &\iff \text{there exists some } v \in U^{\mathcal{A}_{[y/u]}} \text{ such that } \mathcal{A}_{[x/u][z/v]} \models G' \\ &\overset{\text{IH}}{\iff} \text{there exists some } v \in U^{\mathcal{A}_{[y/u]}} \text{ such that } \mathcal{A}_{[y/u][z/v]} \models G'[x/y] \\ &\iff \mathcal{A}_{[y/u]} \models G[x/y] \end{aligned}$$

- $G = \exists x G'$. Then

$$\begin{aligned} \mathcal{A}_{[x/u]} \models G &\iff \text{there exists some } v \in U^{\mathcal{A}_{[x/u]}} \text{ such that } \mathcal{A}_{[x/u][x/v]} \models G' \\ &\iff \text{there exists some } v \in U^{\mathcal{A}_{[x/u]}} \text{ such that } \mathcal{A}_{[x/v]} \models G' \\ &\iff \text{there exists some } v \in U^{\mathcal{A}_{[y/u]}} \text{ such that } \mathcal{A}_{[x/v]} \models G' \\ &\overset{\text{IH}}{\iff} \text{there exists some } v \in U^{\mathcal{A}_{[y/u]}} \text{ such that } \mathcal{A}_{[y/v]} \models G'[x/y] \\ &\iff \text{there exists some } v \in U^{\mathcal{A}_{[y/u]}} \text{ such that } \mathcal{A}_{[y/u][y/v]} \models G'[x/y] \\ &\iff \mathcal{A}_{[y/u]} \models G[x/y] \end{aligned}$$

- $G = \forall z G'$, where $z \neq x$. Can be proven analogously as the $\exists$ case above.
- $G = \forall x G'$. Can be proven analogously as the $\exists$ case above.