Technische Universität München (I7)
Prof. Dr. Dr. h.c J. Esparza / M. Luttenberger / R. Neumann

Summer term 2011
Tuesday 31st May, 2011

Solution

Logic – Homework 3

Discussed on 30.05.2011.

Exercise 3.1

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

(a) Draw the BDDs of F and G.

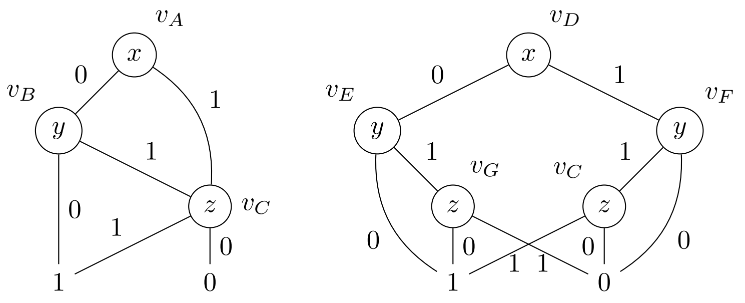
(b) During the lecture an algorithm implementing the OR-operation on BDDs has been presented. Use this algorithm to construct a BDD for $F \vee G$.

(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.

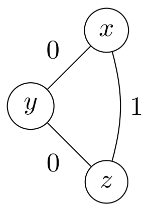
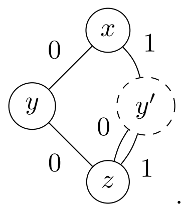
Solution:

(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).



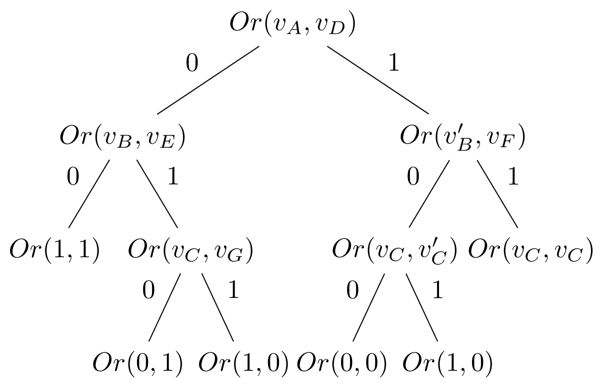
The nodes are assigned names v_x for some x, which are used in task b). Note that v_C has been assigned twice as it indeed represents the same node (if F and G would be in one multiBDD, it would only occur once).

(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

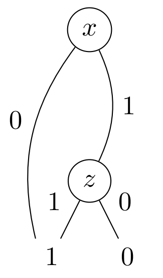
 gets changed to  .

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:

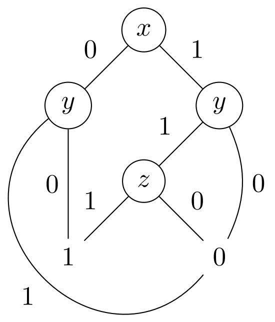


(c) The only things, which need a change are the special cases at the beginning of the algorithm:

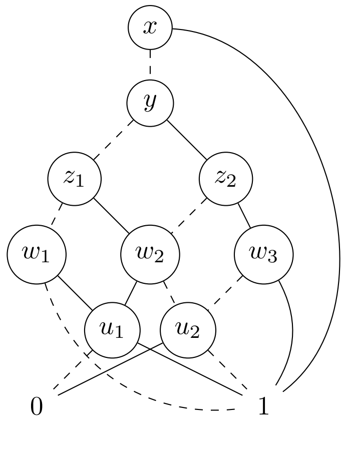
```
**if** $v_F = K_0$ **or** $v_G = K_0$ **then return** $K_0$
**else if** $v_F = v_G = K_1$ **then return** $K_1$
```

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 3.2

There exists a variant of BDDs that quite often produce smaller graphs: *BDDs with complement edges* (CBDDs). An example for such a CBDD can be found further down. The main differences between a normal BDD and a CBDD are as follows:

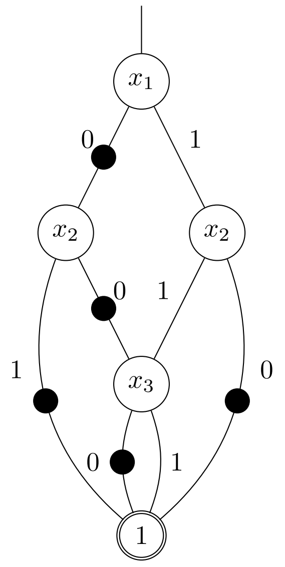
- In a CBDD, there are two kinds of edges:

  Postive (normal) edges, drawn as usual; and negative (complement) edges, explicitly marked by a black dot.
- The root of the CBDD has an incoming edge (which can be negative).
- In a CBDD there is no node labeled by 0.

A negative edge tells us to take the complement of the formula represented by the node the edge points to:

For instance, consider the node labeled by x_3 in the CBDD depicted below: its two children both represent the formula $\top$, but as the edge to its 0-child is negative, the node represents the formula $\neg x_3 \wedge \neg\top \vee x_3 \wedge \top \equiv x_3$.

For another example, consider the left of the two nodes labeled by x_2: this node represents the formula $\neg x_2 \wedge \neg x_3 \vee x_2 \wedge \neg\top \equiv \neg x_2 \wedge \neg x_3$.



(a) State the formula represented by the CBDD above. Also give its truth table.

(b) CBDDs in general are not unique, i.e. for one formula there might be multiple CBDDs representing that formula. Find two formulas that can be represented by at least two CBDDs each. Also state these CBDDs.

(c) If one omits negative 0-edges, it is possible to create unique CBDDs. Therefore show, that for a function $ite(x, F, G) \equiv x \wedge F \vee \neg x \wedge G$ (*if then else*) the following equivalence holds:

$$ite(X, F, \neg G) \equiv \neg ite(x, \neg F, G)$$

Use this equivalence to transform the example CBDD from above into a unique one, which does not contain negative 0-edges.

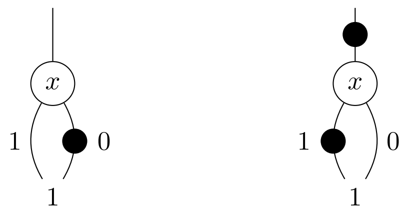
Solution:

(a) The formula is $(x_1 \land x_2 \land x_3) \lor (\neg x_1 \land (x_2 \lor x_3))$

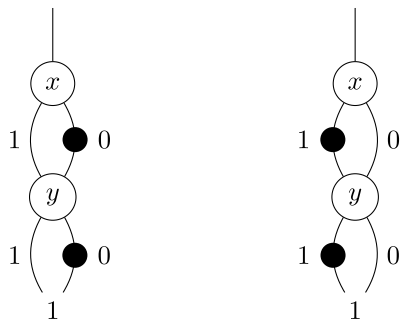
The truth table is:

x_1	x_2	x_3	
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

(b) The formula x:



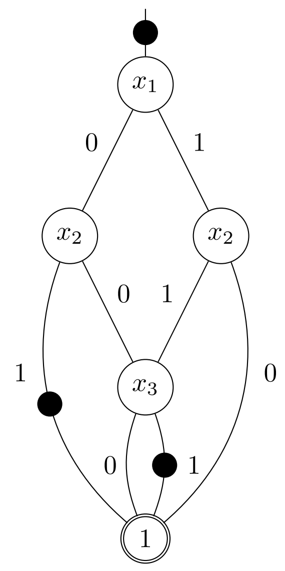
The formula $x \leftrightarrow y$:



(c) $ite(x, F, \neg G)$
$\equiv x \land F \lor \neg x \land \neg G$
$\equiv \neg\neg(x \land F \lor \neg x \land \neg G)$
$\equiv \neg((\neg x \lor \neg F) \land (x \lor G))$
$\equiv \neg((\neg x \land x) \lor (\neg x \land G) \lor (\neg F \land x) \lor (\neg F \land G))$
$\equiv \neg((\neg x \land G) \lor (x \land \neg F))$ *(convince yourself, that the clause $\neg F \land G$ can be safely omitted)*
$\equiv \neg ite(x, \neg F, G)$

This equivalence shows, that if a node has a negated 0-edge, it is possible to use a normal 0-edge, if we negate the 1-edge and the whole formula (i.e. negate all incoming edges). This is done bottom-up in the CBDD and thus yielding a CBDD without negated 0-edges. Please note, that the check whether to apply this equivalence to a node has to be done *after* all of its children have been handled. This is because the application also touches the edge going to a node and thus may modify the out-going edges of the parent.

The unique 0-edge free CBDD of the task hence looks like:


x_1
0
1
x_2
x_2
0
1
1
x_3
0
0
1
1