Algorithm 1 Algorithm for performing $f + g \pmod 2$

```
1:  procedure mod_2_sum(f, g)
2:      if f = 0 then
3:          return g
4:      else if g = 0 then
5:          return f
6:      else if f = g then
7:          return 0
8:      else
9:          v1 = top_var(f); v2 = top_var(g);
10:         if index(v1) < index(v2) then
11:             return ite(v1, then(f), else(f) + g)
12:         else if index(v1) > index(v2) then
13:             return ite(v2, then(g), else(g) + f)
14:         else
15:             return ite(v1, then(f) + then(g), else(f) + else(g))
```

ZBDD Representation: The following steps describe the procedure for building ZBDDs for the polynomials of the gates of circuits.

1. Obtain the RTTO for the variables (signals) of the circuits as $x_1 > x_2 > \cdots > x_n$.

2. Impose the same order on the ZBDDs. When declaring the variables, a unique index is associated with each variable. The index number starts from 0 and increases as we go towards the last variable in the order.

3. Declare ZBDDs for each of these variables. Each of these ZBDDs contains only the root node which the variable itself, and the two terminal nodes with the 1-edge (0-edge) of root going into 1-terminal (0-terminal). The child node of the root at the solid edge's end will be referred to as *then* and the other child as *else*.

4. Use Eqn. (**??**) to model the gates of the circuit as Boolean polynomials. Build ZBDDs for these polynomials using the $+$ and $\cdot$ binary operations for modulo 2 sum and product of variables. The $f + g \pmod 2$ operation can be implemented as $f + g = f_s \cup g_s - f_s \cap g_s$, where f_s and g_s represent the unate cube sets for the polynomials f and g respectively. For example, let $f = ab + c$ and $g = c + d$ with the corresponding unate cube sets $f_s = \{ab, c\}$ and $g_s = \{c, d\}$, then $f_s \cup g_s = \{ab, c, d\}$ and $f_s \cap g_s = \{c\}$. The set difference $f_s \cup g_s - f_s \cap g_s$ is the set $\{ab, d\}$ and the corresponding Boolean polynomial is $ab + d$.

   However, analysis from experiments shows that the union $f_s \cup g_s$ can introduce a large number of terms (large ZBDD size) which are eventually canceled when performing the set difference. In order to avoid the large ZBDD size for the union operation we have implemented the $f + g \pmod 2$ operation as presented in [**?**]. The algorithm for this operation is shown in Algorithm 1.

   In the above algorithm, the function top_var returns the root variable for the input ZBDD (f or g). The function ite is an if-*then-else* operator used for constructing new ZBDDs. The algorithm first checks the simple corner cases in the beginning

of the algorithm and then depending on the index values of the variables (which in our case is RTTO) it recursively constructs new ZBDDs using the *ite* function.

In the above example, let the ordering imposed on the variables is $a > b > c > d$ and the index values for these variables are $0, 1, 2, 3$ respectively. For the given f and g *i.e.* $ab + c$ and $c + d$ respectively, the condition $index(v_1) < index(v_2)$ is true. The *ite* operation places $then(f) = b$ on the solid edge of the root of the new ZBDD and performs $else(f) + g \pmod 2$ for the dotted edge's end as shown in the Fig. 1. During this recursive call, the last condition is true (as $index(v_1) = index(v_2) = 2$). This time the *ite* operation performs two recursive calls $then(f) + then(g)$ and $else(f) + else(g)$. The recursive call $then(f) + then(g)$ returns 0 as both of them are 1 whereas the recursive call $else(f) + else(g)$ returns d as $else(f) = 0$ and $else(g) = d$. Therefore *ite* operation creates a ZBDD with root c $(= v_1 = v_2)$ and its 1-*child* pointing to 0 and 0-*child* pointing to d. Due to the ZBDDs' reduction rules, it gets simplified to just d. Therefore, the resultant ZBDD of the operation $(ab+c)+(c+d)$ has a as its root, 1-*child* as b, and 0-*child* as d while representing the boolean polynomial $ab + d$.

A similar recursive algorithm is also implemented for $f \cdot g \pmod 2$ operation where the intermediate partial product terms are added using the Algorithm 1.

5. Traversing only the solid edges from the root node of a ZBDD to terminal **1** delivers the leading monomial of that polynomial.

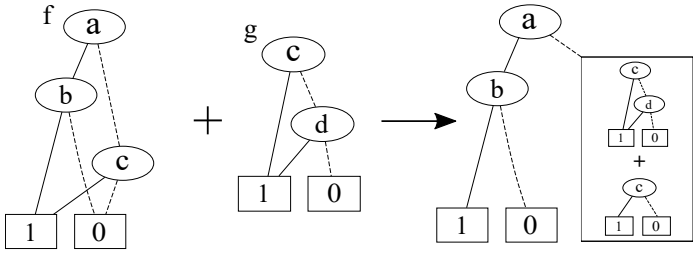

Figure 1: $f + g \pmod 2$ using ZBDDs