Grőbner Basis Computation for Boolean Rings Using Zero Suppressed Binary Decision Diagrams

Revanth Rajalbandi, Harsha Gadiraju and Ian Noy
Dept. of Electrical and Computer Engineering, University of Utah
revanth.rajalbandi@utah.edu, harsha.gadiraju@utah.edu, ian.noy@utah.edu

***Abstract*— Zero Suppressed Binary Decision Diagrams (ZBDDs) are an efficient way to represent and manipulate sets of sets problems. As an ideal is a set of polynomials and each polynomial is a set of monomials, ZBDDs can manipulate these sets and yield faster results when compared to traditional BDDs used in equivalence checking of circuits. ZBDDs encode the functionality of the circuit and so does the Grőbner basis. Here the overall idea is to implement Grőbner basis using ZBDDs with the help of CUDD package (CUDD is a BDD package written in C which implements BDDs and ZBDDs) [1]. This paper describes the algorithm of Grőbner basis computation for boolean rings using Zero Suppresed Binary Decision Diagrams and three cases to test the implementation. The results achieve the desired reduced Grőbner basis forms.**

I. Introduction

Zero-suppressed BDDs are a new type of BDD adapted for representing sets of combinations. They are based on the following reduction rules.

- Eliminate all nodes with the 1-edge pointing to the 0-terminal node. Then connect the edge to the other subgraph directly.
- Share all the equivalent sub-graphs in the same manner as with the conventional BDDs.

Notice that, contrary to conventional BDDs, we do not eliminate nodes whose two edges both point to the same node. This reduction rule is asymmetric for the two edges as the nodes remain when their 0-edge points to a terminal node. When the number and order of the variables are fixed, 0-suppressed BDDs provide canonical forms of Boolean Functions.

Here, we document our attempt at implementing the Buchberger's algorithm on ZBDDs. This is not an original idea, the same type of research has been performed in the PolyBori program [2]. PolyBori provides a Grőbner basis framework for Boolean polynomials. This paper introduces the concept of generating Grőbner basis computations on ZBDDs. The authors also use the CUDD package to manipulate ZBDDs. The purpose of their research is to provide a solution to the problem which is formal verification. There are many ways to verify circuits and running the Buchberger's algorithm on polynomials is an efficient solution. ZBDDs are simplified sets of combinations that can represent polynomials. Combining these ideas into a general model could provide a faster and more efficient means towards formal circuit verification.

II. CUDD Package

Colorado University Decision Diagram (CUDD) is a software package that was developed by Fabio Somenzi at the University of Colorado at Boulder [1]. The package is capable of manipulating BDDs, Algebraic Decision Diagrams (ADDs) and ZBDDs with many sets of operations. These objects can also be converted from one form to another in a seamless manner. When building a ZBDD it is important to treat the function as a set. Here is an example of how a ZBDD is built in CUDD package.

```
// First create a DdManager object
// Next a node representing 1 must be built
DdNode* one = Cudd_ReadZddOne(manager, 0);
// From here a list of variables can be assigned
for(int i = 0, i<3, i++)
    x[i] = Cudd_zddChange(manager, one, i);
    Cudd_Ref(x[i]);
```

The code above creates three ZBDD variables that can represent x, y, and z. From here, there are several functions that can be performed on the variables to generate sets of polynomials.

ZBDDs support many operations for sets of combinations. Some of the basic operations that can be performed are shown here. In the following, **P** and **Q** indicate the instances

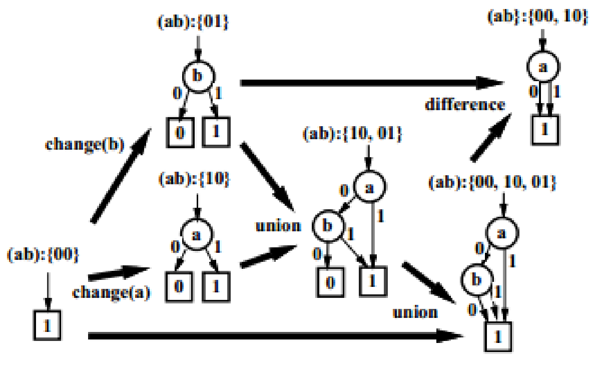

Fig. 1: Construction of ZBDDs for sets of combination [3]
.

of sets of combinations represented by ZBDDs and **v** means an input variable. The flowchart shown in Figure 1 is the method in which the polynomials can be built. Here the polynomials are treated as sets. Any set of combinations can be generated by a sequence of these basic operations. In ZBDDs there is no support for the unary complement operation but it is essential in OBDDs. In case there is a need to compute the complement we first define a universal set **U** to compute complement set $\bar{P}$ and then use the difference operation $(\mathbf{U}-\bar{P})$ to compute it [3].

"Ø"	the empty set. (0-terminal node.)
"**1**"	the set of only the null combination $\{00\ldots0\}$. (1-terminal node.)
P.change(v)	inverts v on each combination.(swaps onset and offset of P.)
$P\cup Q$	the union set of P and Q.
$P\cap Q$	the intersection set of P and Q.
$P-Q$	the difference set.(combinations in P but not in Q.)

III. Algorithms

In order to run Buchberger's algorithm, we need to formulate helper functions. These functions have to be able to process ZBDDs and return the correct object. The first algorithm is to find leading terms in a ZBDD. The algorithm loops through the ZBDD object by traversing the "then" child (also known as the 1 path). The function returns a pointer to a DdNode that represents the leading terms of the ZBDD function.

```
Inputs: $DdManager$, $ZBDD$, $VariableList[\,]$, 1
Outputs: $Leading\ Terms$
1: $temp \leftarrow 1$;
2: int $i = indexofZBDD$;
2: while ($ZBDD \neq$ Const. Node) do
3:   $temp = VariableList[i] * temp$
4:   $ZBDD = then\ child\ of\ ZBDD$
5:   $i = i+1$
6: end while
7: return $temp$;
```

The next algorithm is multivariate division. Again, this function had to be adapted to return a pointer to a DdNode that represents the remainder of the division as a ZBDD. The ZBDD input is the dividend and the ZBDDlist is the divisor. The ZBDDlist is a struct that includes a DdNode pointer and a pointer to the next DdNode in the list. The algorithm loops through each DdNode in the list, finds the leading term and then attempts to divide the ZBDD.

```
Inputs: $DdManager$, $ZBDD$, $ZBDDlist[\,]$, $VariableList[\,]$, 1
Outputs: $Remainder$
1: $zero \leftarrow 0$; $quot \leftarrow 0$; $rem \leftarrow 0$;
2: int $i = 1$;
3: while ($ZBDD \neq zero$) do
4:   if $\exists i$ s.t. $\frac{LT(ZBDD)}{LT(ZBDDlist[i])}$ then
5:     $quot = \{\left(quot \cup \frac{LT(ZBDD)}{LT(ZBDDlist[i])}\right) - \left(quot \cap \frac{LT(ZBDD)}{LT(ZBDDlist[i])}\right)\}$
6:     $ZBDD = \{ZBDD\} - \left\{\frac{LT(ZBDD)}{LT(ZBDDlist[i])}\right\} * ZBDDList[i]$
7:   else
8:     $rem = (rem \cup LT(ZBDD)) - (rem \cap LT(ZBDD))$
9:     $ZBDD = \{ZBDD\} - \{LT(ZBDD)\}$
10:  end if
11:  $i = i+1$;
12: end while
13: return $rem$;
```

One of the key steps in Buchberger's algorithm is finding the s-polynomials of the functions. S-polynomials are formed to cancel the leading terms of two polynomials. The Grőbner basis can be directly defined by these s-polynomials.

Theorem *Let $I \subset K[x_1,x_2,\ldots,x_n]$ be a polynomial ideal with basis $G=\{g_1,g_2,\ldots,g_n\}$ then G is a Grőbner basis for I if and only if the remainder on dividing every $S-$polynomial $S(g_i,g_j)\ (i\neq j)$ by G (in any order of the basis elements) is zero.*[4]

Here is the algorithm for computing s-polynomials on ZBDDs.

```
Inputs: $DdManager$, $ZBDD1$, $ZBDD2$, $VariableList[\,]$, 1
Outputs: $S-polynomial$
1: $LCM = LT(ZBDD1) * LT(ZBBD2)$
2: $prod1 = \frac{LCM}{LT(ZBDD1)} ZBBD1$
3: $prod2 = \frac{LCM}{LT(ZBDD2)} ZBBD2$
4: $prod3 = prod1 \cap prod2$
5: if $prod1 \geq prod2$ then
6:   $diff = \{prod2\} - \{prod3\}$
7:   $sply = \{prod1\} - \{prod2\}$
8:   $sply = (sply \cup diff) - (sply \cap diff)$
9: else
10:  $diff = \{prod1\} - \{prod3\}$
11:  $sply = \{prod2\} - \{prod1\}$
12:  $sply = (sply \cup diff) - (sply \cap diff)$
13: end if
14: return $sply$;
```

With the help of the above mentioned functions, Buchberger's algorithm can be run on ZBDDs. The function below repeatedly computes the s-polynomials of pairs of ZBDDs and then divides by the list of ZBDDs. If the remainder is non-zero then the remainder is added to the list of ZBDDs. The algorithm is shown below.

```
Inputs: $F = List\ of\ ZBDDs$
        $[f_1, f_2, \dots, f_s]$
Outputs: $G = List\ of\ ZBDDs\ representing\ the\ Gr\H{o}bner\ basis$
        $[g_1, g_2, \dots, g_t]$
 1: Initialize: $G := F$; $\mathscr{G} := \{\{f_i, f_j\} \mid f_i \neq f_j \in G\}$
 2: while $\mathscr{G} \neq \emptyset$ do
 3:   Pick a pair $\{f, g\} \in \mathscr{G}$
 4:   $\mathscr{G} := \{\mathscr{G}\} - \{\{f, g\}\}$
 5:   $Spoly(f,g) \overset{G}{\to} +\ h$
 6:   if $h \neq 0$ then
 7:     $\mathscr{G} := \mathscr{G} \cup \{\{u, h\} \mid \forall u \in G\}$
 8:     $G := (G \cup h) - (G \cap h)$
 9:   end if
10: end while
```

Once the Gr\H{o}bner basis has been computed, further simplification and elimination of terms can be performed. The minimal form of a Gr\H{o}bner basis has a leading coefficient of 1 in each of the functions representing the basis. Since we are only working with variables and no coefficients this step is not necessary in our computations. The second step in finding the minimal form is to remove sets from the basis that contain the same leading terms. This can be performed by looping through all pairs of leading terms and deleting if they are equal.

The final step in computing the Gr\H{o}bner basis is to reduce the set of functions. The basis can be further reduced by performing multivariate division on each function in the basis. The algorithm is described below.

```
Inputs: $G = List\ of\ ZBDDs\ representing\ the\ minimal\ Gr\H{o}bner\ basis$
        $[g_1, g_2, \dots, g_s]$
Outputs: $R = List\ of\ ZBDDs\ representing\ the\ reduced\ Gr\H{o}bner\ basis$
        $[r_1, r_2, \dots, r_t]$
1: $R \leftarrow 0$
2: $\forall g_i, g_j \in G$ where $i \neq j$
3: $remainder = \frac{g_i}{g_j}$
4: if $\{remainder \neq 0\}$ then
5:   $R = R \cup remainder$
6: end if
7: return $R$
```

IV. The Experiment

We set out to perform Buchberger's algorithm to ZBDDs. The purpose was to test if we could properly find the Gr\H{o}bner basis of a polynomial that was represented by a ZBDD. Three experiment sets were performed to test the functionality of our code.

A. Case 1

Cyclic Polynomials:

$$f_1 = x_1 + x_2 + x_3 + x_4$$
$$f_2 = x_1x_2 + x_2x_3 + x_3x_4 + x_4x_1$$
$$f_3 = x_1x_2x_3 + x_2x_3x_4 + x_3x_4x_1 + +x_4x_1x_2$$
$$f_4 = x_1x_2x_3x_4 - 1$$

Cyclic polynomials were used to test the code. Here the order maintained is the lexicographic order with (x1,x2,x3,x4). The reduced Gr\H{o}bner basis generated from the output of our code are in the form of minterms { X000 1 , 0X00 1, 00X0 1, 000X 1 } which are $x_1 + 1, x_2 + 1, x_3 + 1, x_4 + 1$ when compared to the input polynomials. Figure 2 shows the ZBDD representation of the reduced Gr\H{o}bner basis obtained using the GraphViz tool.

B. Case 2

More Polynomials:

$$f_1 = ab + 1$$
$$f_2 = ab + a + 1$$
$$f_3 = ab + b + 1$$
$$f_4 = ab + a + b$$
$$f_5 = xy + x + y$$
$$f_6 = yz + y + z$$

We also tested the code with the use of another set of polynomials with the lexicographic order (x,y,z,a,b). The reduced Gr\H{o}bner basis generated from the code are in the form of minterms 00000 1. Figure 3 shows the ZBDD

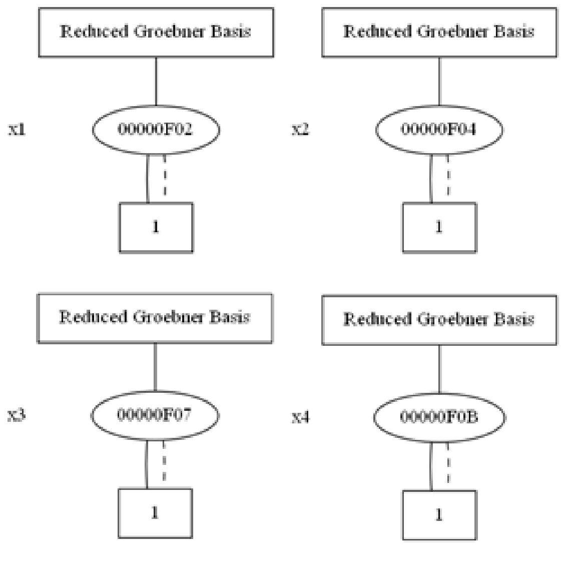

Fig. 2: ZBDD representation of reduced Gr\H{o}bner Basis for cyclic polynomials.

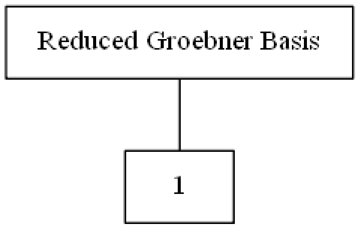

Fig. 3: ZBDD representation of reduced Grőbner Basis for more polynomials.

representation of reduced Grőbner basis obtained using the GraphViz tool.

C. Case 3

Chain of OR Gates:

$$f_1 = z + he + h + e$$
$$f_2 = h + gd + g + d$$
$$f_3 = g + f + c + cf$$
$$f_4 = f + a + b + ab$$

In arithmetic circuits with a chain of OR gates (Figure 4), the reduction process is a big problem. Reduction leads to the generation of a huge number of new monomials and the process takes a long time to complete. We tried reduction (our multivariate division in Singular) of z with the set of equations f_1, f_2, f_3, f_4. It took 10ms in Singular and it took 1ms with our c code.

V. Problems Faced and its Potential Solutions

A few problems were encountered while writing and testing the ZBDD code. Initially we declared the ZBDD variables by using the function cudd_ZddIthvar, but we were not able to perform basic operations such as union and intersection properly. We also had the problem of getting extra nodes while generating ZBDDs. Later, after discussing with the professor about the problem, we understood that we needed to look at ZBDDs as sets. This eliminated all the problems and all the operations such as difference, intersection were working fine. After running several sets of polynomials everything was found to be working fine. When we tried using the cyclic polynomials we found that we had a bug in the code relating to the union operation. We wanted to use the union operation as an addition operation and therefore expected the result of 001 ∪ 001 to be zero, instead the result was 001. Since we are operating in the Boolean domain we had to modify the union operation to get zero. We had to replace the union operation in terms of the intersection i.e.,

$$A + B = (A \cup B) - (A \cap B)$$

and also we were using the inbuilt cudd_ZddunateProduct which internally uses the union operation so we modified it accordingly. With the help of these two steps bug was successfully removed.

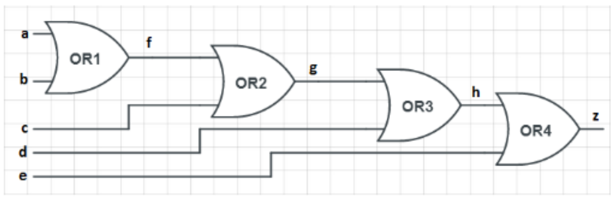

Fig. 4: Top level model of chain of OR gates.

VI. Conclusion

ZBDDs are an efficient method to solve formal verification. In terms of size in memory, ZBDDs are a simplified version of BDDs therefore taking less system space. IV-C shows that we were also able to generate reduction in a more timely fashion. As demonstrated, the buchberger's algorithm can be modified to work on ZBDDs to find the Grőbner basis on the sets of combinations. With a little more time and a program that can generate ZBDDs from functions, we believe that our code generates the correct Grőbner basis for sets of ZBDDs.

VII. Contributions

Harsha Gadiraju was involved in the writing of the major part of the code such as the successful creation of lists using the linked lists, figuring the leading term, S-polynomial and Buchbergers algorithm. Revanth Rajalbandi was involved in the writing of the reduced and minimal Grőbner Basis, helped Harsha while writing the code, researching the information out of the Journal papers and debugging the code by solving the Grőbner basis manually on the paper. Ian Noy was involved in figuring out the problems in the multivariate division, conversion of leading terms and multivariate division into functions and he spent a lot time in our initial attempts to create ZBDDs.

Other contributions come from the information gathered from scholarly resources. Shin-ichi Minato wrote three very good papers on BDDs and ZBDDs [3], [5], [6]. He is regarded as one of the key researchers in ZBDDs.

References

[1] F. Somenzi, "Cudd: Cu decision diagram package release 2.5. 0," *University of Colorado at Boulder*, 1998.

[2] M. Brickenstein and A. Dreyer, "Polybori: A framework for gröbner-basis computations with boolean polynomials," *Journal of Symbolic Computation*, vol. 44, no. 9, pp. 1326–1345, 2009.

[3] S. ichi Minato, "Zero-suppressed bdds and their applications," *International Journal on Software Tools for Technology Transfer*, vol. 3, no. 2, pp. 156–170, 2001.

[4] J. Selig, "S-polynomials and buchberger's algorithm," *London South Bank University*, London SE1 0AA, UK.

[5] S. ichi Minato, "Zero-suppressed bdds for set manipulation in combinatorial problems," in *Design Automation, 1993. 30th Conference on*. IEEE, 1993, pp. 272–277.

[6] ——, *Binary Decision Diagrams and Applications for VLSI CAD*. Springer, 1996, vol. 342.