Implementation of Boolean arithmetic in SAGE math - Report

Akash (160020009), Neil Dalal (16D070014), Sucheta Ravikanti (160040100)
Ghanshyam (16D070014), Anshul Verma (16D070024)

Abstract— **Boolean functions have wide applications in the areas of cryptography due to its simplicity and efficient algorithms for computations. We have made use of the SAGE Math package available on python to implement various operations on boolean polynomials, the usage of which will be discussed.**

I. LITERATURE SURVEY

Traditional methods for solving a system of Boolean equations involves converting the entire system into a single equation and proceeding to solve the SAT problem. [1] resolves it by solving the same using general orthonormal (ON) expansions. It introduces the idea of splitting the problem into smaller sub-problems simplifying the search of solution on the sub-problems and essentially parallelizing the algorithm.

Another such paper that has introduced a novel algorithm in the field of Boolean algebra is [2]. It developed a parallel computational solver that can compute all the satisfying assignments of a Boolean system of equations even when there are many variables. The paper also remarked that this method is very effective in the case of sparse factors in a formulae.

Boolean solvers have been extensively used in Cryptanalysis of various encryption schemes. Many primitives like block ciphers use as building blocks some Boolean functions with several output bits. Such functions are called vectorial Boolean functions, or S-boxes. Owing to the substantial usage of Boolean functions, the Boolean Arithmetic Solver we have implemented will find uses in diverse applications.

II. INTRODUCTION

A boolean function of *n* variables is a function F_2^n into F_2.

The input polynomials will be given in the Algebraic Normal Form (ANF). It is represented as follows, for any Boolean function f of n variables. $x_0, x_1, ..., x_n$ denote the variables and a subscripts denote the coefficients.

$$f(x_0, x_1, ...x_n) = (a_0) \oplus (a_1x_1 \oplus a_2x_2 \oplus ...a_nx_n) \oplus (a_{1,2}x_1x_2 \oplus ...a_{n-1,n}x_{n-1}x_n) \oplus ...(a_{1,2,...,n}x_1x_2...x_n)$$

This is a normal form, meaning that any two equivalent forms will always convert to the same ANF, therefore making this form useful in finding whether any two given forms are equivalent or not. The added advantage of ANF is that it can be represented as a simple list of lists of variables, which we have exploited in our implementation.

Theorem: A boolean function $f : B_o^n \to B_o$ has a unique representation as [3]

$$f = a_0 \oplus \sum_{\forall i}^{\oplus} a_i x_i \oplus \sum_{\forall i,j}^{\oplus} a_{i,j} x_{i,j} \oplus ... \oplus a_{1,2,...,n} x_1 x_2 ... x_n$$

III. NOTATION & RUNNING THE CODE

We have developed code for implementing several boolean arithmetic operations on SAGE Math for 8 variables, which can be easily extended to any variable number of variables. All modules take inputs in ANF and output back in ANF. The operations that fall in our domain are:

- **Addition** : Takes input boolean polynomials f and g that are $f, g : B_o^n \to B_o$ and gives f+g as the output.
- **Product** : Takes input boolean polynomials f and g that are $f, g : B_o^n \to B_o$ and gives f*g as the output.
- **Composition** : Takes input boolean polynomial *h(u, v)* that is, $h : B_o^2 \to B_o$ and *f*, *g* that are $f, g : B_o^n \to B_o$ to output $h(f(x_0, x_1, ..., x_n), g(x_0, x_1, ..., x_n))$.
- **Substitution** : Take input boolean polynomial f that is $f : B_o^n \to B_o$ and an implicant *t* to output the boolean polynomial that remains after substituting $t = 1$ into f that is, $f(t = 1)$.

ANF inputs are expected to be a list of lists of variable names, which is explained below with examples.

$$f = [1,\ [\ [3],\ [4,5]\ ]\ ] = 1 + x_3 + x_4 * x_5$$

$$g = [1,\ [\ [0],\ [4,2]\ ]\ ] = 1 + x_0 + x_4 * x_2$$

$$h(u,v) = [\ 0,\ [\ [0],\ [0,1]\ ]\ ] = u + uv$$

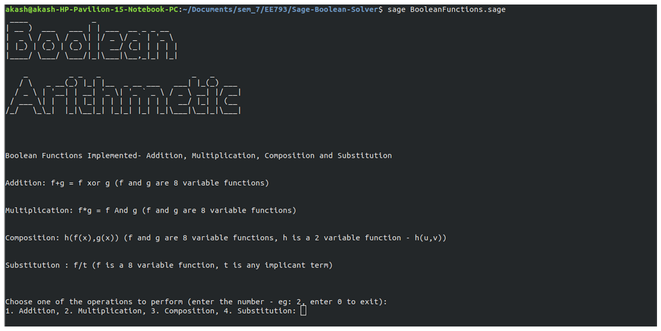
$$t = [0,\ 3,\ 5] = x_0 * x_3 * x_5$$

We have made the code as interactive and user-friendly as possible by explicitly giving examples highlighting the format in which the outputs are expected for the modules to function as expected. Refer Figure 1.

IV. RESULTS

We have tested all the functions by making test cases and comparing the received outputs with the expected expressions. One such test case for each of the functions is shown below, along with the received output.

For further details on running the scripts and dependencies, refer to {Github Link} for the full repository containing all the files.



```
akash@akash-HP-Pavilion-15-Notebook-PC:~/Documents/sem_7/EE793/Sage-Boolean-Solver$ sage BooleanFunctions.sage

Boolean Functions Implemented- Addition, Multiplication, Composition and Substitution


Addition: f+g = f xor g (f and g are 8 variable functions)


Multiplication: f*g = f And g (f and g are 8 variable functions)


Composition: h(f(x),g(x)) (f and g are 8 variable functions, h is a 2 variable function - h(u,v))


Substitution : f/t (f is a 8 variable function, t is any implicant term)



Choose one of the operations to perform (enter the number - eg: 2, enter 0 to exit):
1. Addition, 2. Multiplication, 3. Composition, 4. Substitution: 
```

Fig. 1: Interface of Boolean Solver on Terminal

- **Addition**

  $f = [1,\ [\ [0],\ [3],\ [4,5],\ [6,7],\ [0,1,2,4]\ ]\ ] = 1 + x_0 + x_3 + x_4 * x_5 + x_6 * x_7 + x_0 * x_1 * x_2 * x_4$

  $g = [1,\ [\ [0],\ [3],\ [4,2],\ [6,5],\ [0,2,4]\ ]\ ] = 1 + x_0 + x_3 + x_4 * x_2 + x_6 * x_5 + x_0 * x_2 * x_4$

  $f + g = [0, [[0,1,2,4], [0,2,4], [2,4], [4,5], [5,6], [6,7]]] = x_0 * x_1 * x_2 * x_4 + x_0 * x_2 * x_4 + x_2 * x_4 + x_4 * x_5 + x_5 * x_6 + x_6 * x_7$

- **Product**

  $f = [1,\ [\ [3],\ [4,5]\ ]\ ] = 1 + x_3 + x_4 * x_5$

  $g = [1,\ [\ [0],\ [4,2]\ ]\ ] = 1 + x_0 + x_4 * x_2$

  $f * g = [1, [[0,4,5], [2,3,4], [2,4,5], [0,3], [2,4], [4,5], [0], [3]]] = 1 + x_0 * x_4 * x_5 + x_2 * x_3 * x_4 + x_2 * x_4 * x_5 + x_0 * x_3 + x_2 * x_4 + x_4 * x_5 + x_0 + x_3$

- **Composition**

  $f = [1, [[0], [3], [4,5]]] = 1 + x_0 + x_3 + x_4 * x_5$

  $g = [0, [[0], [3], [4,2]]] = x_0 + x_3 + x_4 * x_2$

  $h(u,v) = [0, [[0], [0,1]]] = u + uv$

  $h(f,g) = [1, [[0,2,4], [0,4,5], [2,3,4], [2,4,5], [3,4,5], [2,4], [4,5], [0], [3]]] = 1 + x_0 * x_2 * x_4 + x_0 * x_4 * x_5 + x_2 * x_3 * x_4 + x_2 * x_4 * x_5 + x_3 * x_4 * x_5 + x_2 * x_4 + x_4 * x_5 + x_0 + x_3$

- **Substitution**

  $f = [1, [[0], [3], [4,5], [6,7], [0,1,2,4]]] = 1 + x_0 + x_3 + x_4 * x_5 + x_6 * x_7 + x_0 * x_1 * x_2 * x_4$

  $t = [0,3,5] = x_0 * x_3 * x_5$

  $f/t = [1, [[1,2,4], [6,7], [4]]] = 1 + x_1 * x_2 * x_4 + x_6 * x_7 + x_4$

V. CONCLUSIONS

This implementation can be easily extended to any number of variables.

ACKNOWLEDGMENT

We would like to thank Prof. Virendra Sule for giving us the required guidance and opportunity to work on this project.

REFERENCES

[1] Virendra Sule, "An algorithm for Boolean satisfiability based on generalized orthonormal expansion", arXiv:1406.4712v3, 2014
[2] Virendra Sule, "Implicant based parallel all solution solver for Boolean satisfiability", arXiv:1611.09590v3, 2016
[3] Topics in Cryptology, Class Notebook