EE793 - Assignment No. 2

M V Girish Srivatsa

S Prajeeth

Himanshu Sheoran

March 2021

Contents

1 Algorithm: **2**
- 1.1 Introduction: . . . 2
- 1.2 Variables in a Boolean polynomial: . . . 2
- 1.3 Computation of Orthonormal Basis: . . . 2
- 1.4 Quotient of a formula with a term: . . . 3
- 1.5 Generate Implicants: . . . 3
- 1.6 Basic operations on Boolean Polynomials: . . . 4
- 1.7 Theory: . . . 4
- 1.8 Recursive based Solver: . . . 4

2 Time Complexity: **5**

3 Verification: **5**

1 Algorithm:

1.1 Introduction:

In this report we show an implementation of an implicant based All-SAT solver for formula's in ANF form,A high level overview of the algorithms and theory used in this report is based on the paper suggested[1].

All notations used in the report shall follow the same.We shall also attempt to refine the algorithm to decrease time complexity and perform fairly efficient solving even for dense polynomials based on higher levels of parallelisation.

We shall also split the algorithm into various components and show various optimizations performed in SageMath[2] to reduce time required.

1.2 Variables in a Boolean polynomial:

We now start of with a primitive implementation of obtaining list of variables in SageMath

```
def primitiveVariables(formula):
    return list(formula.variables())
```

```
sage: %timeit primitiveVariables(f)
1.13ms±19.3µs per loop(mean±std.dev.of 7 run,1000 loops each)
```

Now,on a lookup in the SageMath source code for the variables primitive we observe that it performs a class of input sanitization which when combined with typcasting into tuple and back to list takes time.So an efficient implementation of this can be done by:

```
def variables(formula):
    z = [formula.ring()(i)
    for i in str(formula.vars_as_monomial()).split('*')]
    one = formula.ring().one()
    if one in z:
        z.remove(one)
    return z
```

```
sage: %timeit variables(f)
29.2µs±135ns per loop(mean±std.dev. of 7 runs,10000 loops each)
```

1.3 Computation of Orthonormal Basis:

This is a fairly straightforward part.We simply take the orthonormal basis of the form x0,$\neg x0$ & x1,$\neg x0$ & $\neg x1$,... An optimization that is performed here shall be covered in the part regarding quotient of a formula w.r.t a term

1.4 Quotient of a formula with a term:

We know,

$$f/t = f(t(X) = 1)$$

Based on this we can compute the quotient if we know the assignment for which the term evaluates to 1.For the case of computation of quotient with *orthonormal basis*,we modify the orthonormal basis generation function to compute its assignment map simultaneously.For other cases we can compute the map by the formula,

$$t * x = 0 \iff x + 1 \in t$$

So,if $t * x$ evaluates to 0 we assign x to 0 else to 1 based on this map we compute quotient by a substitution to the polynomial(sage primitive method subs())

1.5 Generate Implicants:

For this we observe the algorithm used in the paper and perform an additional level of optimization(parallelization) to it,

Algorithm 1: GenImplicants

Input : f a Boolean function (or an equation $f = 1$)implicants is to be found
Output: I a complete set of implicants of f (or an equation $f = 1$)

```
1  X = variables(f)
2  ortho = ON(X) // generates an orthonormal basis in variables X
3  I = φ
4  for each t ∈ ortho //start a new thread with (f,t) do
5  |  q =quotient(f,t)
6  |  if q = 1 then
7  |  |  I ← I ∪ t
8  |  else if q = 0 then
9  |  |  goto end
10 |  else
11 |  |  I_temp ←GenImplicants(q)
12 |  |  I ← I ∪ {t * I_temp}
13 |  |  return I
14 |  end
15 |  return I
16 end
17 Gather thread outputs(I),GenImplicants(f) is union of all I
```

This algorithm is more productive in the case of dense Boolean polynomials(since in sparse case the thread depth is small making parallelization overhead large)

We have made attempts at implementing the above algorithm based on the two provided methods given is SageMath(RecursivelyEnumeratedSets,@parallel decorator)

1.6 Basic operations on Boolean Polynomials:

This is simple as we use the fact that in Boolean Polynomial Ring addition is xor operation,multiplication is and operation,for or operation we use the formula,

$$a \| b = a \oplus b \oplus ab$$

These are now designed and implemented as anf_xor,anf_or,anf_and in the code. With regards to a from scratch implementation we have created the same in python based on the formulae being passed in the list based notation for ANF in the paper with the same function names.Their algorithms are as follows:

- anf_not: we only need to take a bitwise xor with the constant term in the expression
- and_xor: we compute the parity of occurence of any term and discard terms occuring even number of times
- anf_and: For this we implement the product of a term against another formula(by simple concatenation and check of parity) and then iteratively multiply term by term
- anf_or: This is computed by the formula mentioned above

1.7 Theory:

We can classify the theory used here under two algorithms,

- GenImplicants:

  We use Proposition 3 of the paper[1] to implement a GenImplicants algorithm

- RecursiveSolve:

  We use Proposition 2 along with the recursive decomposition,

$$F_m = f_1 * F_{m-1}$$

$$F_{m1k} = f_{k+1} * F_{m-k-1}$$

$$F_1 = f_m$$

1.8 Recursive based Solver:

In the case of recursive implicant based solver,we have implemented the algorithm as given in the paper:

With respect to the parallelization we have attempted both methods provided by SageMath.i.e,RecursivelyEnumeratedSet,@parallel decorators and defined them as solve and solve_2 in the code respectively.

With regards to the choice of the pivot we have followed the method given in the paper(choosing formula with minimum number of variables) as the heuristic.

2 Time Complexity:

The time taken for various values are(m = 5),

Total #vars	subset_size	#pairs sampled	Time(solve)*	Time(solve_2)*
5	2	3	103 ms	59.7 ms
5	3	5	104 ms	64.9 ms
5	4	7	140 ms	237 ms
10	4	6	132 ms	164 ms
10	4	8	138 ms	284 ms
10	5	10	235 ms	1.07 s
15	4	10	278 ms	366 ms
15	5	15	248 ms	1.3 s
15	6	20	299 ms	4.68 s
20	5	15	377 ms	3.21 s
20	6	20	304 ms	2.83 s
20	7	20	278 ms	Time exceeded

*all times used are proccess times(sys time+user time)
Here, we have used SAGE_NUM_THREADS=9, which we found it to be optimal on average. Tested on: HP Pavilion dk0051, 12GB RAM, 6 cores, 2 threads per core.

3 Verification:

To verify the solve method we only require that the set returned is both orthogonal and is a complete set of implicants.
Variables used in sample: $x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9$
Functions:

$$f_1 = x_1x_7 + x_2x_7 + x_4x_9 + x_5x_7 + x_7$$

$$f_2 = x_1x_6 + x_1x_9 + x_2x_7 + x_4x_5 + x_4x_7$$

$$F = \prod_{i=1}^{2} f_i$$

F=$x_1x_2x_6x_7 + x_1x_2x_7x_9 + x_1x_2x_7 + x_1x_4x_5x_7 + x_1x_4x_6x_9 + x_1x_4x_7 + x_1x_4x_9 + x_1x_5x_6x_7 + x_1x_5x_7x_9 + x_2x_4x_5x_7 + x_2x_4x_7x_9 + x_2x_4x_7 + x_2x_5x_7 + x_4x_5x_7 + x_4x_5x_9 + x_4x_7x_9 + x_4x_7$
On solving(by solve) we obtain output as:

$$I_1 = x_1x_2x_4x_5x_6x_7x_9 + x_1x_2x_4x_5x_6x_7 + x_1x_4x_5x_6x_7x_9 + x_1x_4x_5x_6x_7$$

$$I_2 = x_1x_2x_4x_5x_6x_7x_9 + x_1x_2x_4x_5x_6x_7 + x_1x_2x_4x_6x_7x_9 + x_1x_2x_4x_6x_7$$

$$I_3 = x_1x_2x_4x_5x_6x_7x_9 + x_1x_2x_4x_6x_7x_9 + x_1x_2x_5x_6x_7x_9 + x_1x_2x_6x_7x_9$$

$I_4 = x_1x_2x_4x_5x_6x_7x_9+x_1x_2x_4x_5x_6x_7+x_1x_2x_4x_5x_7x_9+x_1x_2x_4x_5x_7+x_1x_2x_4x_6x_7x_9+x_1x_2x_4x_6x_7 + x_1x_2x_4x_7x_9 + x_1x_2x_4x_7 + x_1x_2x_5x_6x_7x_9 + x_1x_2x_5x_6x_7 + x_1x_2x_5x_7x_9 + x_1x_2x_5x_7 + x_1x_2x_6x_7x_9 + x_1x_2x_6x_7 + x_1x_2x_7x_9 + x_1x_2x_7$

$$I_5 = x_1x_2x_4x_5x_7 + x_1x_2x_5x_7 + x_2x_4x_5x_7 + x_2x_5x_7$$

$I_6 = x_1x_2x_4x_5x_6x_7x_9 + x_1x_2x_4x_5x_7x_9 + x_1x_2x_5x_6x_7x_9 + x_1x_2x_5x_7x_9 + x_1x_4x_5x_6x_7x_9 + x_1x_4x_5x_7x_9 + x_1x_5x_6x_7x_9 + x_1x_5x_7x_9$

$I_7 = x_1x_2x_4x_5x_6x_7x_9 + x_1x_2x_4x_5x_6x_7 + x_1x_2x_5x_6x_7x_9 + x_1x_2x_5x_6x_7 + x_1x_4x_5x_6x_7x_9 + x_1x_4x_5x_6x_7 + x_1x_5x_6x_7x_9 + x_1x_5x_6x_7$

$I_8 = x_1x_2x_4x_5x_7x_9 + x_1x_2x_4x_5x_7 + x_1x_2x_4x_7x_9 + x_1x_2x_4x_7 + x_1x_4x_5x_7x_9 + x_1x_4x_5x_7 + x_1x_4x_7x_9 + x_1x_4x_7 + x_2x_4x_5x_7x_9 + x_2x_4x_5x_7 + x_2x_4x_7x_9 + x_2x_4x_7 + x_4x_5x_7x_9 + x_4x_5x_7 + x_4x_7x_9 + x_4x_7$

$$I_9 = x_1x_2x_4x_5x_6x_7x_9 + x_1x_2x_4x_5x_6x_9 + x_1x_4x_5x_6x_7x_9 + x_1x_4x_5x_6x_9$$

$$I_{10} = x_1x_2x_4x_5x_6x_9$$

$$I_{11} = x_1x_2x_4x_5x_7x_9 + x_1x_2x_4x_5x_7 + x_2x_4x_5x_7x_9 + x_2x_4x_5x_7$$

$$I_{12} = x_1x_2x_4x_5x_7x_9 + x_1x_2x_4x_5x_9 + x_2x_4x_5x_7x_9 + x_2x_4x_5x_9$$

$I_{13} = x_1x_2x_4x_5x_6x_7x_9 + x_1x_2x_4x_5x_6x_9 + x_1x_2x_4x_5x_7x_9 + x_1x_2x_4x_5x_9 + x_1x_2x_4x_6x_7x_9 + x_1x_2x_4x_6x_9 + x_1x_2x_4x_7x_9 + x_1x_2x_4x_9$

$I_{14} = x_1x_2x_4x_5x_7x_9 + x_1x_2x_4x_5x_9 + x_1x_4x_5x_7x_9 + x_1x_4x_5x_9 + x_2x_4x_5x_7x_9 + x_2x_4x_5x_9 + x_4x_5x_7x_9 + x_4x_5x_9$

$$I_{15} = x_1x_2x_4x_5x_6x_7x_9 + x_1x_2x_4x_6x_7x_9 + x_1x_4x_5x_6x_7x_9 + x_1x_4x_6x_7x_9$$

$I_{16} = x_1x_2x_4x_5x_6x_7x_9+x_1x_2x_4x_5x_6x_9+x_1x_2x_4x_5x_7x_9+x_1x_2x_4x_5x_9+x_1x_2x_4x_6x_7x_9+x_1x_2x_4x_6x_9 + x_1x_2x_4x_7x_9 + x_1x_2x_4x_9 + x_1x_4x_5x_6x_7x_9 + x_1x_4x_5x_6x_9 + x_1x_4x_5x_7x_9 + x_1x_4x_5x_9 + x_1x_4x_6x_7x_9 + x_1x_4x_6x_9 + x_1x_4x_7x_9 + x_1x_4x_9$

We require sum of all implicants to be equal to F,
$\sum_{j=1}^{16} I_j = x_1x_2x_6x_7 + x_1x_2x_7x_9 + x_1x_2x_7 + x_1x_4x_5x_7 + x_1x_4x_6x_9 + x_1x_4x_7 + x_1x_4x_9 + x_1x_5x_6x_7 + x_1x_5x_7x_9 + x_2x_4x_5x_7 + x_2x_4x_7x_9 + x_2x_4x_7 + x_2x_5x_7 + x_4x_5x_7 + x_4x_5x_9 + x_4x_7x_9 + x_4x_7$

We also require all implicants to be orthogonal,we verify this by:

```
sage: def check(I):
....:     for i in I:
....:         for j in I:
....:             if i != j:
```

```
....:                     assert i*j == 0,"ERROR"
sage: check(z)
sage:
```

Since there was no ERROR, we conclude that the assertion was satisfied.So the solve function yields a **complete set of orthogonal implicants**.

References

[1] Virendra Sule. Implicant based parallel all solution solver for boolean satisfiability. Technical report, Indian Institute of Technology Bombay, 02 2017.

[2] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 9.2)*, 2021. `https://www.sagemath.org`.