IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.8, Issue.7, July 2020 E-ISSN: 2347-2693

Unsheathing input to SAT solver from logic circuit in DIMAC format

Vaishnavi Thorve

Dept. of Electronics and Communication, Government Engineering College, Bhavnagar, India

Author's Mail Id: vaishnavithorve710@gmail.com

DOI: https://doi.org/10.26438/ijcse/v8i7.132135 | Available online at: www.ijcseonline.org

Received: 28/June/2020, Accepted: 15/July/2020, Published: 31/July/2020

Abstract **-** Satisfiability probability(SAT) solvers are algorithms that take well-formed formulas and return true if satisfiable and false otherwise. They are significantly useful in various tasks in-circuit verification and have prominent importance in various fields such as automated verification, Electronic Design Automation (EDA) which include formal checking of equivalence, artificial intelligence planning, and so on. The input to these SAT solvers is generally accepted in DIMAC CNF (conjunctive normal form), which is a textual representation of the equation in CNF form. Most verification tasks are commenced from a description of problem instances at the logic circuit level, deducing DIMAC CNF format from the given logic circuit could be tedious and time-consuming if done manually as the conversion includes complex formulas. This paper aims to present an effective code that could easily convert any given logic circuit to DIMAC CNF format just by selecting gate and proper variable names to operate upon according to the heuristic presented by a logic circuit. The output thus formed could be directly given as input to any SAT solver that uses DIMAC format readily, abating time and conversion efforts. The final part of the paper also demonstrates an example for the acyclic circuit and compares the result of generated output to manually derived formula for the same circuit on various parameters.

Keyword**:** DIMAC, CNF, SAT solvers, input generator

I.INTRODUCTION

Satisfiability probability(SAT) solvers work on a propositional formula for a given Boolean expression or well-formed formulas and provide the assignment that conveys whether the given Boolean formula is satisfied. Gomes et al. (1) state that "Despite the worst-case exponential run time of all known algorithms, satisfiability solvers are increasingly leaving their mark as a general-purpose tool in areas as diverse as software and hardware verification, automatic test pattern generation, planning, scheduling and even challenging problems from algebra".

Satisfiability problems often trace back to be computationally inaccessible as it belongs to a Nondeterministic polynomial time(NP) complexity class. However as emphasized by Gilles et al. (2), "with the introduction of lazy data-structures, efficient learning mechanisms, and activity-based heuristics, the picture is quite different. In many combinatorial field, the state-of-the-art schema is often to use an adequate polynomial reduction to this canonical NP-Complete problem, and then to solve it efficiently with an SAT solver", using Boolean constraint Propagation(BCP).

SAT solver formula is usually represented with CNF, and they generally use the Davis-Putnam-Logemann-Loveland (DPLL) algorithm for deciding the satisfiability of propositional logic formula.

Many SAT solvers preferably accept the input in DIMAC CNF format, which is a textual representation of CNF formula. Zhaohui et al. (3) stated "Most circuit verification tasks start from a logic circuit description of the problem instance", favorably there are conversion heuristics from logic circuit network to CNF form, but the process of extracting appropriate input from the logic circuit can be extremely time-consuming as each gate present in the logic circuit will have to be converted to its CNF form with formulas given in table 1. A problem instance description taken from the real world could include as many as 50,000 variables and solve for each gate individually could be a daunting and tedious task, so the correct approach would be to formulate a way to extract this format efficiently and with little work.

II. MOTIVATION FOR RESEARCH

This section provides an overall motivation for carrying out the research. First, the dearth of effective measures on finding a solution for the unsheathing of DIMACS which leads to time-consuming conversions carried by individuals when the real motive is distress. Second, the ubiquity of the DIMACS in any problem-solving in SAT solvers will incorporate such a method. Third, many researchers mainly focus on finding new ways to solve the existential problem, but this research will bridge the existing present method to bring about the required changes.

The aim of this paper is thus to provide an efficient code for the generation of the proper DIMAC format input for a given logic circuit, by selecting or giving appropriate gates and variables as input to the code. This will thus in turn provide an output, which could be readily given to any

SAT solvers as input and thus finds the required satisfiability assignments of variables.

The work presented by Klimek et al. (4), is based on "random generation of long and complex logical formulas", it does not include the participation of unsheathing formulas from logic circuits which could be the basic form of information and problem instance. Gopalakrishnan et al. (5) represent work with the integration of Binary Decision Diagrams(BDD's) and CNF, while the main focus of this paper will be entirely on CNF format. Zhaohui et al. (3) presents an overall query about the loss of structural information while using circuit-based verification and provides an adequate result to bridge the gap by applying heuristics which could be of great use, thus dynamically approaching the problems.

This research could thus provide a great methodology to unsheathing the inputs that could be used on a wide range of works related to AI planning to the normal curriculum of students to find the desired solution by giving them a quick method to determine the inputs for there SAT solvers. It will be useful in finding solutions with the knowledge present today, giving it a quick upgrade to be useful to a large group of people where this could be applied in the real world.

Conjunctive normal form (CNF)

Not many problems and expressions we encounter in our day to day lives are in CNF format, hence conversion is often an essential part while formulating CNF.

CNF is a set of clauses consisting of literals, which are "AND of OR" representation of Boolean variables. Smith (6) inferred "The CNF formula representing the constraints for a gate can be generated from characteristic function', and can be converted to a minimal constraint using Karnaugh map".

Smith (6) "This process can be repeated for any gate type, including multi-output gates. The construction will be polynomial in the number of inputs and outputs as long as "counting"-type gates are assumed to have a fixed maximum number of inputs".

Table 1 gives the CNF for all the gate types which could be satisfactorily used in converting logic gates into CNF form.

The C code also makes use of these formulas for formulating CNF and forming the required output. Smith (6) stated, "Any variable assignment that satisfies this formula directly yields a valid assignment of logic values to circuit lines".

III.DIMAC CNF FORMAT AND NOTATION

DIMAC format is one of the prevalent formats vividly used by most of the SAT solvers and they give a textual representation of CNF in ASCII characters.

DIMAC generally consists of a preamble which depicts the information that the format holds within each line. In the given format every subsequent row starts with a variable that describes the type of the line.

1] comments: this particular line starts with the variable 'c' and is used to depict additional information about the file which is human readable, and the program ignores this line.

2] Problem line: these lines start with the variable 'p', each problem instance has this line and it depicts the FORMAT which is expected**,** VARIABLES in the instance and CLAUSE present in the instance, and follows the structure shown below:

p FORMAT VARIABLES CLAUSES
p: indicates the start of the problem line.
FORMAT: determines the expected format by the problem, and contains the term "cnf".
VARIABLES: depicts the number of variables present in a particular instance.
CLAUSE: depicts the number of clauses present.
Each subsequent line followed by the structure of the problem line gives the variable in natural number representation and their assignments present within the clause. The numbers in each row may or may not include a "-" minus sign that precedes the number, this sign represents a negation of the variable. The zero in the rows marks the end of that particular row.

1. EXAMPLE: the given CNF formula
(g | -h | i) ^ (-j) ^ (-h | i)
Is given in DIMAC format as follows

```
c files
p cnf 4 3
1 -2 3 0
-4 0
```

IV. CODE AND ITS WORKING

The aim of this code is to generate an input sequence that could be readily given to the SAT solvers. The important aspect of this code is its effectiveness in deducing the formula in minimum time with apt parameterization.

It takes into account all the preambles of DIMAC format and thus provides the required output. The approach used in this is solely based on circuit verification and thus is a unique approach for solving satisfiability problems.

The formula generation process involves giving appropriate variables and gates as an input and can be done as follows:

Keyword: DIMAC, CNF, SAT solvers, input generator

Let us initially consider a part of the logic circuit inferred from a real-time application circuit and find the output for the same in DIMAC CNF format so as to check its satisfiability.

1] Figure 1.1 shows a logic circuit for which this code will be used to find the required output. The conversions done in this program are based on the formulas which are given in table1.

Table 1. CNF FORMULAS FOR ELEMENTRY GATES

GATE	FUNCTION	CNF FORMULA
AND	y=x1 . x2 .....xn.	$(\prod_{i=1}^{n}(xi+\bar{y}))(\sum_{i=1}^{n}\overline{xi}+y)$
NAND	$\overline{\text{y=x1 . x2 .....xn.}}$	$(\prod_{i=1}^{n}(xi+y))(\sum_{i=1}^{n}\overline{xi}+\bar{y})$
OR	y=x1 + x2 + ... +xn.	$(\prod_{i=1}^{n}(\overline{xi}+y))(\sum_{i=1}^{n}xi+\bar{y})$
NOR	$\overline{\text{y=x1 + x2 + ... +xn.}}$	$(\prod_{i=1}^{n}(\overline{xi}+\bar{y}))(\sum_{i=1}^{n}xi+y)$
XOR	$y=x1\oplus x2$	$(\overline{x1}+\overline{x2}+\bar{y})(x1+x2+\bar{y})$ $(\overline{x1}+x2+y)(x1+\overline{x2}+y)$
NXOR	$\overline{y=x1\oplus x2}$	$(\overline{x1}+\overline{x2}+y)(x1+x2+y)$ $(\overline{x1}+x2+\bar{y})(x1+\overline{x2}+\bar{y})$
BUFFER	y=x	$(x+\bar{y})(\bar{x}+y)$
NOT	$\overline{\text{y = x}}$	$(x+y)(\bar{x}+\bar{y})$
MUX	$y=(s\leftrightarrow 1)?x1:x2$	$(\overline{x1}+s+y)(x1+s+\bar{y})$ $(x2+\bar{s}+\bar{y})(\overline{x2}+\bar{s}+y)$
HALF-ADD	$s=a\oplus b$ $cout=a\bullet b$	$(a+\overline{cout})(b+\overline{cout})$ $(a+b+\bar{s})(\bar{a}+\bar{b}+\bar{s})$ $(\bar{a}+cout+s)(\bar{b}+cout+s)$
FULL-ADD	$s=a\oplus b\oplus c$ $cout=(a\oplus b)\bullet cin+ab$	$(a+b+\overline{cout})(\bar{a}+\bar{b}+cout)$ $(a+cin+\overline{cout})(\bar{a}+\overline{cin}+cout)$ $(b+cin+\overline{cout})(\bar{b}+\overline{cin}+cout)$ $(s+\bar{a}+cout)(\bar{s}+a+\overline{cout})$ $(s+\bar{b}+cout)(\bar{s}+b+\overline{cout})$ $(s+\overline{cin}+cout)(\bar{s}+cin+\overline{cout})$ $(s+\bar{a}+\bar{b}+\overline{cin})(\bar{s}+a+b+cin)$

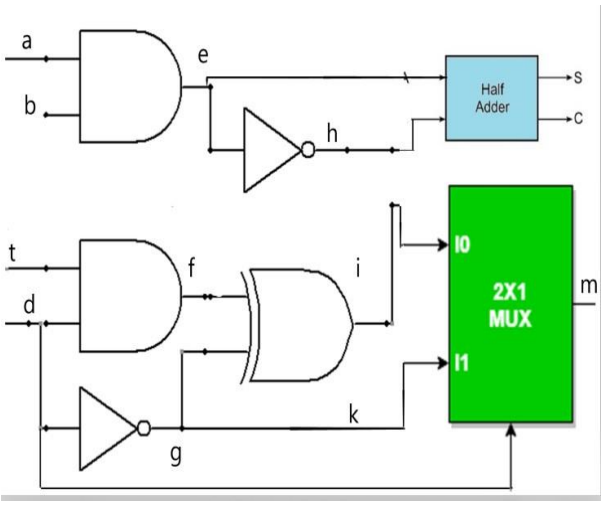

Figure 1.1: Logic Circuit

2] Figure 2 shows the screenshot of the initial program which gives the user the notation that is being used to depict a particular gate in the instance and the user has to enter this notation according to the gates the problem involves.

This also gives the users the benefit to enter multiple gates of their choice or to find an output after each gate is given its required input.

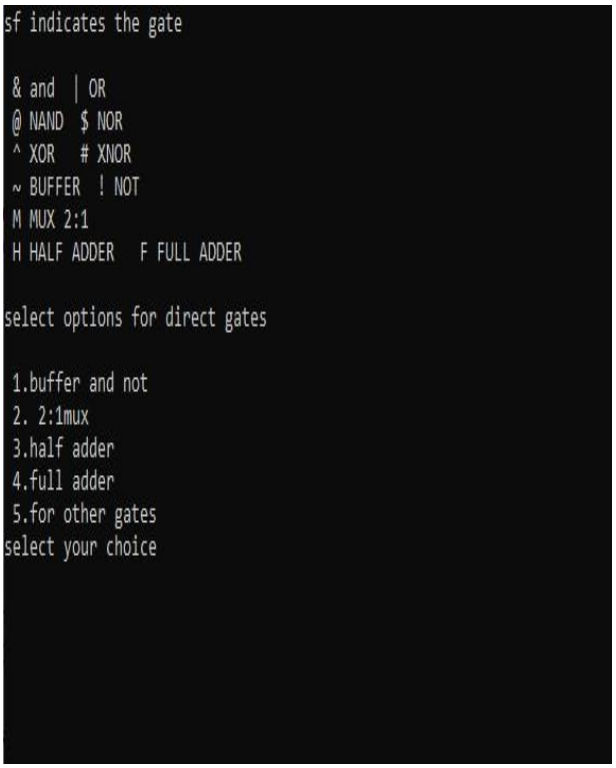

Figure 2: Introduction to Code Notations

3]Figure 3 shows how the code is provided with the essential variables, which are the inputs the gate takes in a logic circuit format.


```
select options for direct gates

 1.buffer and not
 2. 2:1mux
 3.half adder
 4.full adder
 5.for other gates
select your choice 5
sf according to gates described above
enter ip1 a

enter ip2 b

enter op e

enter sf &

enter  1.for new iput and 2.To print final output1


select options for direct gates

 1.buffer and not
 2. 2:1mux
 3.half adder
 4.full adder
 5.for other gates
select your choice 5
```

Figure 3: Input To The Code

4]Figure 4 thus gives the final output for the entire logic circuit shown in figure 1. This output can be copied and thus be given to any SAT solver that takes this input format and thus see the satisfiability.

```
output
p cnf 13 24
 1 -3 0
 2 -3 0
-1 -2 -3 0
 4 -6 0
 5 -6 0
-4 -5 -6 0
 5 7 0
 -5 -7 0
 3 8 0
 -3 -8 0
 3 -10 0
 8 -10 0
 3 8 -9 0
 -3 -8 -9 0
 -3 10 9 0
 -8 10 9 0
 -6 -7 11 0
 6 7 11 0
 -6 7 11 0
 6 -7 11 0
 -11 5 13 0
 11 5 -13 0
 12 -5 -13 0
 -12 -5 13 0
```

Figure 4: Output Of Logic Circuit

V. RESULTS

In the above-carried work, figure 4 we can see the final output of a simple logic circuit having 7 gates, this obtained result is now directly fed into MiniSAT solver and the final desired results are shown in figure 5.

The solver accepts the input given to it which is the direct result that was obtained in figure 4 and no potential changes are needed to be made int the output before feeding it into any SAT solver.

```
MiniSAT Results
SAT
-1 -2 -3 -4 -5 -6 7 8 9 -10 11 -12 13 0
MiniSAT Standard output
WARNING: for repeatability, setting FPU to use double precision
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:           13                                          |
|  Number of clauses:             24                                          |
|  Parse time:                  0.00 s                                        |
|  Eliminated clauses:          0.00 Mb                                       |
|  Simplification time:         0.00 s                                        |
|                                                                             |
============================[ Search Statistics ]==============================
```

Figure 5: Output From Sat Solver

The above figure thus gives the results returning the satisfiable conditions for the given logic circuit in figure 1.1. It tells what inputs are to be given at each input so as to obtain the desired result which could be easily traced and implemented in the original circuit.

The output received after successful implementation of the code is compared by manual conversion and the output thus obtained is similar to the result obtained by pen and paper, this gives an overall idea about the effectiveness and accuracy of the code used. The output that is inferred can now be used directly as an input to various SAT solvers including MiniSAT, MaxSAT, etc, which are the widely used solvers in today's ti

VI. CONCLUSION AND FUTURE SCOPE

In this paper, the main focus is given to form a DIMAC CNF formula from a particularly given logic circuit, the main idea is to focus on the inputs that are to be generated to give to any SAT solver rather than focusing on the satisfiability. Then it goes on to produce a dynamic code that could help to formulate the final inputs.

The core limitations are the lack of memory space to hold on to large inputs given to the code, this could be further solved by creating a website thus making sure the memory size is expanded and the user can input any desirable amount of logic gates thus to find the required result and work conveniently on a larger platform.

REFERENCES

[1] C Gomes, H Kautz, A Sabharwal, B Selman. "Satisfiability Solvers"2008.https://www.cs.cornell.edu/gomes/pdf/2008_gomes_knowledge_satisfiability.pdf

[2] A.Gilles, S Laurent. "Predicting Learnt Clauses Quality in Modern SAT Solvers". IJCAI International Joint Conference on Artificial Intelligence., pp **399-404 2009.**

[3] F Zhaohui, Y Yinlei, M Sharad ." Considering Circuit Observability Don't Care in CNF Satisfiability". DATE'05, Mar 2005, Munich, Germany. pp.**1108-1113**. ?hal-00181279?

[4] R Klimek, K Grobler-Dębska, E Kucharska . "System for automatic generation of logical formulas". MATEC Web of Conferences.,pp **252**:**03005**,**2019**;.

[5] S Gopalakrishnan, V Durairaj, P Kalla. "Integrating CNF and BDD based SAT solvers". Eighth IEEE International High-Level Design Validation and Test Workshop; **2003**.

[6] A Smith ." Diagnosis of combinational logic circuits using Boolean satisfiability". Ottawa: Library and Archives Canada = Bibliothèque et Archives Canada; **2005**.

[7] N Eén, N Sörensson ." Translating Pseudo-Boolean Constraints into SAT". Journal on Satisfiability, Boolean Modeling, and Computation. pp **2(1-4):1-26**.,**2006**.

[8] Ifat Jahangir, Anindya Das, Masud Hasan, "Facile Algebric Representation of a Novel Quaternary Logic".International Journal of Computer Sciences and Engineering Vol.**4** Issue 5,pp **1-15**, **2016**.

AUTHORS PROFILE

Vaishnavi Thorve, an undergraduate student pursuing B.Tech in Electronics and Communication Engineering from Government Engineering College Bhavnagar, Gujarat India.Interested field of study VLSI and computer science coding, and their tie-up in bringing about automotive changes.

