Oracle-Guided Incremental SAT Solver User manual

Duo Liu, Cunxi Yu, Xiangyu Zhang, Daniel Holcomb

Software User Manual

version 0.0.0, 04 Nov 2015

Abstract

This document is the Software User Manual (SUM) for the Oracle-Guided Incremental SAT Solver (Solver). The Software User Manual (SUM) instructs how to install and use the Oracle-Guided Incremental SAT Solver (Solver).

Contents

1 Introduction

1.1 Purpose

Solving camouflage circuit is a notoriously NP problem. The Oracle-Guided Incremental SAT Solver (Solver) is specified in solving camouflage circuit with extremely high efficiency (5 times faster than existing best solver).

1.2 Principle

Solver executes a loop that continually finds new input and output vectors using SAT queries and an oracle circuit model. After some number of iterations, the constraints accumulated are sufficient to rule out all logical functions except for the one that is the true function of the obfuscated circuit. Detailed description please refer to citation.

1.3 Terminology

- Oracle circuit: original circuit without any obfuscated gate.
- Camouflage circuit: obfuscated *oracle* circuit.
- allowed bits: possible solution, known in advance, for camouflage circuit
- forbidden bits: complementary set of allowed bits

2 Tutorial

2.1 Dependencies

- minisat-incre-simp: main program.
- decam-incre.py, genMtrs.py, grabNodes.py: parser, translate combinational circuits to CNF clauses by way of Tseitin encoding
- completeset.py, testforbid.py: library, used to assign forbidden bits

2.2 Initialization

Makefile is included in directory, use command below to initialize working environment.

$ make

2.3 Command

After initializing, *solver* can be accessed from command line:

$ ./minisat-incre-simp decam-incre <oracle.v > <camouflage.v >

- *oracle*.v : input *oracle* circuit path
- *camouflage*.v : input *camouflage* circuit path

For example, if the oracle circuit is "c432-abcmap-fmt.v" and the camouflage circuit is "c432-mux4-101.v", then the command should be:

$ ./minisat-incre-simp decam-incre c432-abcmap-fmt.v c432-mux4-101.v

2.4 Sample input format

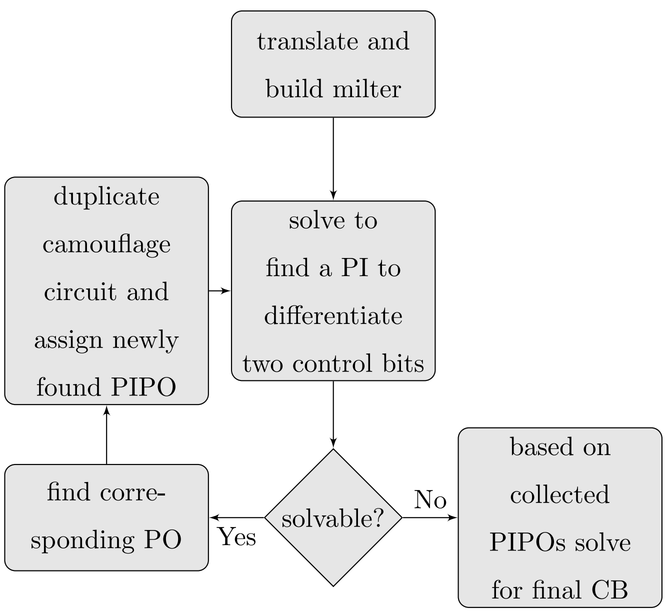
Solver only takes ABC format netlist, the following is an example:

```
module c17 (N1,N2,N3,N4,N5,N6,N7,N8,N9,N12,N13,N14);
input N1,N2_new,N3,N4,N5  //RE__PI;
input X_1 //RE__ALLOW(0,1);
input p1,p2,p3,p4     //RE__ALLOW(1,0);
output N6,N7;
wire N9,N12,N13,N14,N3_NOT,N4_NOT,EX1,EX2,EX3,EX4,EX5,EX6,EX7,EX8,EX9,
EX10,EX11,N2;
nand2 gate1( .a(N1), .b(N3), .O(N14) );
inv1 gate( .a(N3),.O(N3_NOT) );
inv1 gate( .a(N4),.O(N4_NOT));
and2 gate( .a(N3_NOT), .b(p1), .O(EX2) );
and2 gate( .a(N4_NOT), .b(EX2), .O(EX3) );
and2 gate( .a(N3), .b(p2), .O(EX4) );
and2 gate( .a(N4_NOT), .b(EX4), .O(EX5) );
and2 gate( .a(N3_NOT), .b(p3), .O(EX6) );
and2 gate( .a(N4), .b(EX6), .O(EX7) );
and2 gate( .a(N3), .b(p4), .O(EX8) );
and2 gate( .a(N4), .b(EX8), .O(EX9) );
or2  gate( .a(EX3), .b(EX5), .O(EX10) );
or2  gate( .a(EX7), .b(EX10), .O(EX11) );
or2  gate( .a(EX9), .b(EX11), .O(N9) );
nand2 gate3( .a(N2), .b(N9), .O(N13) );
nand2 gate4( .a(N5), .b(N9), .O(N12) );
nand2 gate5( .a(N14), .b(N13), .O(N6) );
nand2 gate6( .a(N12), .b(N12), .O(N7) );
xor2 gate( .a(X_1), .b(N2_new), .O(N2) );
endmodule
```

There are several points need attention:

- format is based on ABC format.
- primary input should be noted by //RE__PI.
- control bits should be noted by //RE__ALLOW.
- allowed bits for each group of control bits should be written after //RE__ALLOW, for example //RE__ALLOW(1,0). If one camouflage gate requires more than one bit, for example it needs two bits, use the format //RE__ALLOW(00,01,10,11) alternatively.

2.5 Call Tree



3 History Version

- version 0.0.0, 04 Nov 2015

4 Contact

5 Citation

- *Oracle-Guided Incremental SAT Solving to Reverse Engineer Camouflaged Logic Circuits*