Department of Electrical Engineering
National Taiwan University
Introduction to EDA, Spring 2018

May 14, 2018
Iris Hui-Ru Jiang

Programming Assignment #2: Equivalence Checking

Due 11:59pm, May 28, 2018

1. Introduction

Equivalence checking is an essential task in functional verification. In this assignment, we study the problem of equivalence checking (EC) for two combinational circuits. The EC problem can be formulated as a CNF verified by SAT.

2. Problem statement

2.1. Brief description

Write a program to generate the CNF for verifying the equivalence of two combinational circuits. As discussed in our lectures, a miter is introduced to compare circuit outputs. As shown in Fig. 1, two combinational circuits C1 and C2 are equivalent if and only if the output of their "miter" structure always produces constant 0.

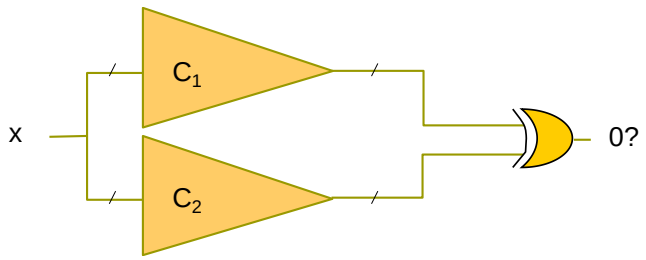


Fig. 1. Miter structure.

To complete the CNF, we translate each combinational gate to several clauses based on Tseitin transformation as listed in Fig. 2.

Type	Operation	CNF Sub-expression
AND	$C = A \cdot B$	$(\overline{A} \vee \overline{B} \vee C) \wedge (A \vee \overline{C}) \wedge (B \vee \overline{C})$
NAND	$C = \overline{A \cdot B}$	$(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee C) \wedge (B \vee C)$
OR	$C = A + B$	$(A \vee B \vee \overline{C}) \wedge (\overline{A} \vee C) \wedge (\overline{B} \vee C)$
NOR	$C = \overline{A + B}$	$(A \vee B \vee C) \wedge (\overline{A} \vee \overline{C}) \wedge (\overline{B} \vee \overline{C})$
NOT	$C = \overline{A}$	$(\overline{A} \vee \overline{C}) \wedge (A \vee C)$
XOR	$C = A \oplus B$	$(\overline{A} \vee \overline{B} \vee \overline{C}) \wedge (A \vee B \vee \overline{C}) \wedge (A \vee \overline{B} \vee C) \wedge (\overline{A} \vee B \vee C)$

Fig. 2. Tseitin transformation.

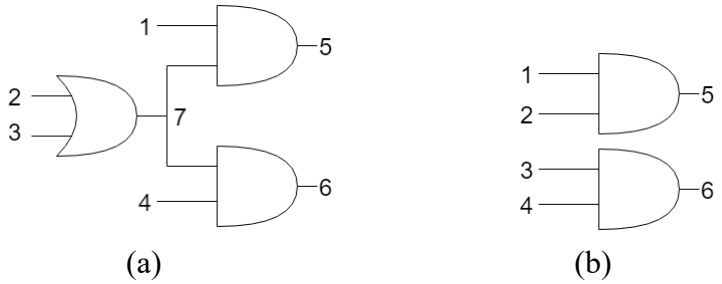
2.2. Input/output specification

The binary should be named as ec. The program should be invoked like this:

```
./ec [*_A.bench] [*_B.bench] [*.dimacs]
```

- Input

Two input bench files describes two combinational circuits. The sample input files given in Fig. 3 are as follows.



(a) (b)

Fig. 3. Two input combinational circuits. (a) Circuit A. (b) Circuit B.

Sample input

Circuit A:

```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
OUTPUT(5)
OUTPUT(6)
7 = OR(2, 3)
5 = AND(1, 7)
6 = AND(4, 7)
```

Circuit B:

```
INPUT(1)
INPUT(2)
INPUT(3)
INPUT(4)
OUTPUT(5)
OUTPUT(6)
5 = AND(1, 2)
6 = AND(3, 4)
```

- Output

The program generates the corresponding CNF of the miter structure as shown in Fig. 4.

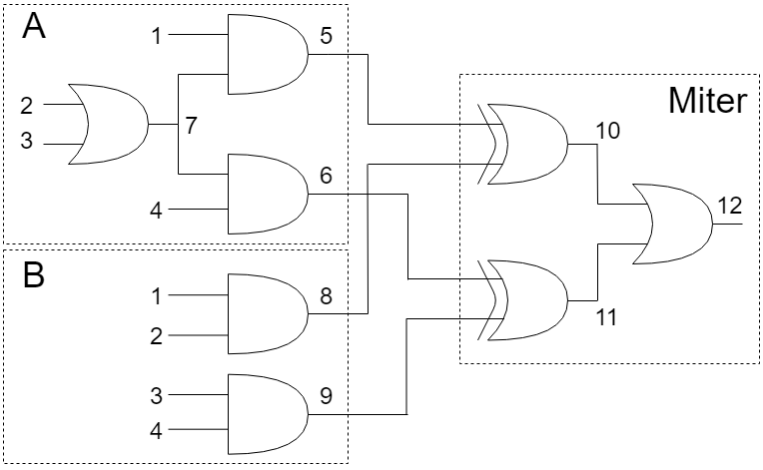


Fig. 4. The miter structure of circuits A and B.

Sample output

The CNF for the given example should be:

```
p cnf 12 27
1 -5 0
7 -5 0
5 -1 -7 0
4 -6 0
7 -6 0
6 -4 -7 0
-2 7 0
-3 7 0
-7 2 3 0
1 -8 0
2 -8 0
8 -1 -2 0
3 -9 0
4 -9 0
9 -3 -4 0
-5 -8 -10 0
5 8 -10 0
5 -8 10 0
-5 8 10 0
-6 -9 -11 0
6 9 -11 0
6 -9 11 0
-6 9 11 0
-10 12 0
```

```
-11 12 0
-12 10 11 0
12 0
```

The generated CNF will be solved by MiniSat, and the solution will be checked based on its correctness. Fig. 5 shows the log of solving the above CNF by MiniSat. The CNF is satisfiable, i.e., circuits A and B are not equivalent.

```
jjwang@eda33 ~/Teaching_Assistant/EDA18/PA2/PA2_mit $ ./MiniSat_v1.14_linux example.dimacs
============================[MINISAT]===================================
| Conflicts |     ORIGINAL     |              LEARNT              | Progress |
|           | Clauses Literals |   Limit Clauses Literals  Lit/Cl |          |
==========================================================================
|         0 |      24       66 |       8       0        0     nan |  0.000 % |
==========================================================================
restarts              : 1
conflicts             : 0              (nan /sec)
decisions             : 3              (inf /sec)
propagations          : 12             (inf /sec)
conflict literals     : 0              ( nan % deleted)
Memory used           : 1.67 MB
CPU time              : 0 s

SATISFIABLE
```

Fig. 5. The snapshot of log generated by MiniSat.

3. Evaluation

Your program should be able to handle thousands of gates. Each case is individually evaluated by correctness. If the binary takes more than **1 minute** to generate CNF, this case will be graded fail even if the results are correct.

4. References

[1] MiniSat v1.14. http://minisat.se/

5. Submission

- ATTENTION: Plagiarism MUST be avoided. Every submission will be tested by a plagiarism catcher, running on all submissions from this class. If plagiarism is discovered (similarity ≥ 40%), all students involved will receive only partial credit. Specifically, if the score received is x and there are n students involved, then the score for each student is x/n.
- Compress the source code, binary code, and a readme file to a zip file named as: StudentID_pa1.zip (e.g., b03912345_pa2.zip) Please briefly describe how to compile and execute your code in readme.
- Submit the zip file to ceiba