Binary Decision Diagram Implementation

ECE1733 Topics in Switching Theory

Zeming Li (1001654845)
Yuchen Wang (1002821574)

Abstract

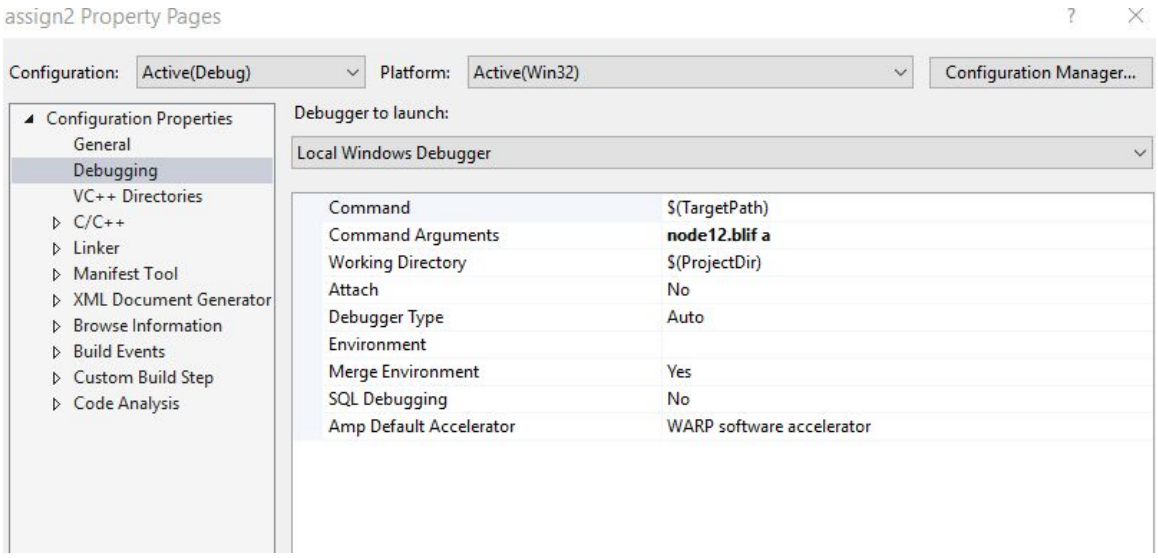
In this assignment, we write a C program to to implement of a binary decision diagram (BDD) from input BLIF files that describe logic functions. Features of the program includes to build the BDD, perform sifting algorithm on BDD and apply Boolean operation between two BDDs. The program also measures the runtime of apply algorithm and sifting algorithm. In part IV, the CUDD package provides basic functions to manipulate Binary Decision Diagrams (BDDs), and a program is created for the apply algorithm based on the CUDD package. The runtime for apply function implemented with CUDD is compared with our own algorithm.

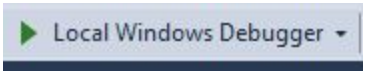
Implementation

This program was written in C language with Microsoft Visual studio. There are two ways to implement this program, one is to run the solution package, the other is to directly run .exe file.

- Run Solution Package

The complete visual studio solution is submitted. To run the program, download the zip file and unzip it to a destination folder. Find the assign2.sln file in the directory and open it with Microsoft Visual Studio. Edit solution property and input test BLIF file and operation command (i.e. apply, build or sifting) in “command arguments window”



Lastly, run the program by clicking Load Windows Debugger 

.

- Run executable file

An executable file (assign2.exe) was generated and can be directly run in command prompt.

a) Download the file ***assign2*** to a folder of choice and put test node files (.blif) under the same directory. To test build and sifting function, the test node file must contain one function. To test apply function, the test node file must contain two functions.

b) Rename file name to ***assign2.exe***

c) Open command prompt and use change directory command (cd) to the destination folder. The command prompt will run assign1.exe and the result will be printed in the same command window screen.

d) Testing

- Test Build function： Input command "***assign2.exe <source BLIF file> build"*** (e.g. assign1.exe node1.blif build) to execute assign2 with input data in node1.blif
- Test Sifting function： Input command "***assign2.exe <source BLIF file> sifting"*** (e.g. assign1.exe node1.blif sifting) to execute assign2 with input data in node1.blif
- Test Apply function： Input command "***assign2.exe <source BLIF file> apply"*** (e.g. assign1.exe node12.blif apply) to execute assign2 with input data in node12.blif (there are two functions in node12.blif)

- CUDD Implementation:

Download CUDD package and unzip it to a destination folder. We implemented apply algorithm based on the main function of nanotrav package in CUDD. Compiling this package follows the same method as compiling nanotrav package. Specifically, refer to the following commands.

- Change directory to folder cudd_3.0.0/nanotrav and input the following command: ***"execute ./nanotrav <file1.blif> -second <file2.blif> -apply <operation>"***

(e.g.: ./nanotrav a.blif -second b.blif -apply xnor) to execute and operation for two logic functions.

Test Results

Note:

- The second Var column represents input variable index (i.e. 0 means X1, 1 means X2, 3 means X3, 4 means X4, or a/b/c/d respectively)
- 0 path/low path : the next node when selecting the current variable = 0
- 1 path/high path : the next node when selecting the current variable = 1

Problem 1: build package tested on node2.blif

```
*****************************************
ECE 1733 Assignment 2
Zeming Li and Yuchen Wang
*****************************************
Reading blif file node2.blif...

Function g has inputs:
Input x1
Input x2
Input x3
Input x4

====================================
|Node   |Var    |0 Path | 1 Path
|0      |
|1      |
|2      |3      |1      |0
|3      |1      |2      |0
|4      |2      |1      |2
|5      |3      |0      |1
|6      |2      |0      |5
|7      |1      |4      |6
|8      |0      |3      |7
====================================
Original BDD for Function g.....
Graph g has 4 inputs and 9 nodes in total
Graph g {
```

Fig: Table for function g in node2.blif

```
Graph g {
        8 [label="x1"];
        7 [label="x2"];
        6 [label="x3"];
        5 [label="x4"];
        5 -> 1 [High Path];
        5 -> 0 [Low Path];
        6 -> 5 [High Path];
        6 -> 0 [Low Path];
        7 -> 6 [High Path];
        4 [label="x3"];
        2 [label="x4"];
        2 -> 0 [High Path];
        2 -> 1 [Low Path];
        4 -> 2 [High Path];
        4 -> 1 [Low Path];
        7 -> 4 [Low Path];
        8 -> 7 [High Path];
        3 [label="x2"];
        3 -> 0 [High Path];
        8 -> 3 [Low Path];
        0 [Terminal Box];
        1 [Terminal Box];
}

Done.
Press any key to continue . . .
```

Fig: graph for function g in node2.blif

Problem 2: apply algorithm tested on node12.blif

```
Select C:\Users\Zeming\Desktop\Assign2_new\Debug\assign2.exe
*********************************************
ECE 1733 Assignment 2
Zeming Li and Yuchen Wang
*********************************************
Reading blif file node12.blif...

Function f has inputs:
Input x1
Input x2
Input x3
Input x4

====================================
|Node   |Var    |0 Path | 1 Path
|0      |
|1      |
|2      |3      |1      |0
|3      |2      |0      |2
|4      |1      |1      |3
|5      |3      |0      |1
|6      |2      |5      |0
|7      |2      |0      |5
|8      |1      |6      |7
|9      |0      |4      |8
====================================
Original BDD for Function f.....
Graph f has 4 inputs and 10 nodes in total
Graph f {
```

Fig: Table for function f in node12.blif

```
Original BDD for Function f.....
Graph f has 4 inputs and 10 nodes in total
Graph f {
        9 [label="x1"];
        8 [label="x2"];
        7 [label="x3"];
        5 [label="x4"];
        5 -> 1 [High Path];
        5 -> 0 [Low Path];
        7 -> 5 [High Path];
        7 -> 0 [Low Path];
        8 -> 7 [High Path];
        6 [label="x3"];
        6 -> 0 [High Path];
        8 -> 6 [Low Path];
        9 -> 8 [High Path];
        4 [label="x2"];
        3 [label="x3"];
        2 [label="x4"];
        2 -> 0 [High Path];
        2 -> 1 [Low Path];
        3 -> 2 [High Path];
        3 -> 0 [Low Path];
        4 -> 3 [High Path];
        4 -> 1 [Low Path];
        9 -> 4 [Low Path];
        0 [Terminal Box];
        1 [Terminal Box];
}

Function g has inputs:
Input x1
Input x2
Input x3
```

Fig: Graph for function f in node12.blif

```
Function g has inputs:
Input x1
Input x2
Input x3

====================================
|Node   |Var    |0 Path | 1 Path
|0      |
|1      |
|10     |2      |1      |0
|11     |2      |0      |1
|12     |1      |10     |11
|13     |0      |1      |12
====================================
Original BDD for Function g.....
Graph g has 3 inputs and 6 nodes in total
Graph g {
        13 [label="x1"];
        12 [label="x2"];
        11 [label="x3"];
        11 -> 1 [High Path];
        11 -> 0 [Low Path];
        12 -> 11 [High Path];
        10 [label="x3"];
        10 -> 0 [High Path];
        10 -> 1 [Low Path];
        12 -> 10 [Low Path];
        13 -> 12 [High Path];
        13 -> 1 [Low Path];
        0 [Terminal Box];
        1 [Terminal Box];
}
```

Fig: Table and graph for function g in node12.blif

```
Enter two functions and operation command or "exit":
Apply>f and g

Resulting BDD:
Time taken for apply operation = 0.007931 ms
Graph f_and_g {
        9 [label="x1"];
        8 [label="x2"];
        7 [label="x3"];
        5 [label="x4"];
        5 -> 1 [High Path];
        5 -> 0 [Low Path];
        7 -> 5 [High Path];
        7 -> 0 [Low Path];
        8 -> 7 [High Path];
        6 [label="x3"];
        6 -> 0 [High Path];
        8 -> 6 [Low Path];
        9 -> 8 [High Path];
        4 [label="x2"];
        3 [label="x3"];
        2 [label="x4"];
        2 -> 0 [High Path];
        2 -> 1 [Low Path];
        3 -> 2 [High Path];
        3 -> 0 [Low Path];
        4 -> 3 [High Path];
        4 -> 1 [Low Path];
        9 -> 4 [Low Path];
        0 [Terminal Box];
        1 [Terminal Box];
}

Apply boolean operation for two BDD diagrams
Enter two functions and operation command or "exit":
Apply>
```

Fig :operation time and result graph for f and g

```
Enter two functions and operation command or "exit":
Apply>f or g

Resulting BDD:
Time taken for apply operation = 0.013063 ms
Graph f_or_g {
        13 [label="x1"];
        12 [label="x2"];
        11 [label="x3"];
        11 -> 1 [High Path];
        11 -> 0 [Low Path];
        12 -> 11 [High Path];
        10 [label="x3"];
        10 -> 0 [High Path];
        10 -> 1 [Low Path];
        12 -> 10 [Low Path];
        13 -> 12 [High Path];
        13 -> 1 [Low Path];
        0 [Terminal Box];
        1 [Terminal Box];
}

Apply boolean operation for two BDD diagrams
Enter two functions and operation command or "exit":
Apply>
```

Fig: operation time and result graph for f or g

Problem 3: sifting algorithm tested on node2.blif

```
*****************************************
ECE 1733 Assignment 2
Zeming Li and Yuchen Wang
*****************************************
Reading blif file node2.blif...

Function g has inputs:
Input x1
Input x2
Input x3
Input x4

====================================
|Node   |Var    |0 Path | 1 Path
|0      |
|1      |
|2      |3      |1      |0
|3      |1      |2      |0
|4      |2      |1      |2
|5      |3      |0      |1
|6      |2      |0      |5
|7      |1      |4      |6
|8      |0      |3      |7
====================================
Original BDD for Function g.....
Graph g has 4 inputs and 9 nodes in total
Graph g {
```

Fig: Table for function g in node2.blif

```
Graph g has 4 inputs and 9 nodes in total
Graph g {
        8 [label="x1"];
        7 [label="x2"];
        6 [label="x3"];
        5 [label="x4"];
        5 -> 1 [High Path];
        5 -> 0 [Low Path];
        6 -> 5 [High Path];
        6 -> 0 [Low Path];
        7 -> 6 [High Path];
        4 [label="x3"];
        2 [label="x4"];
        2 -> 0 [High Path];
        2 -> 1 [Low Path];
        4 -> 2 [High Path];
        4 -> 1 [Low Path];
        7 -> 4 [Low Path];
        8 -> 7 [High Path];
        3 [label="x2"];
        3 -> 0 [High Path];
        8 -> 3 [Low Path];
        0 [Terminal Box];
        1 [Terminal Box];
}

Sifting operation starts:
Total Swap Operation = 16


Post Sifting Table
====================================
|Node   |Var    |0 Path | 1 Path
|0      |
|1      |
|2      |3      |1      |0
|3      |1      |2      |0
|4      |2      |1      |2
|5      |3      |0      |1
|6      |2      |0      |5
|7      |1      |4      |6
|8      |0      |3      |7
====================================
```

Fig: graph for function g in node2.blif and post sifting table

```
BDD after sifting operation.....
Graph g has 4 inputs and 9 nodes in total
Graph g {
        8 [label="x1"];
        7 [label="x2"];
        6 [label="x3"];
        5 [label="x4"];
        5 -> 1 [High Path];
        5 -> 0 [Low Path];
        6 -> 5 [High Path];
        6 -> 0 [Low Path];
        7 -> 6 [High Path];
        4 [label="x3"];
        2 [label="x4"];
        2 -> 0 [High Path];
        2 -> 1 [Low Path];
        4 -> 2 [High Path];
        4 -> 1 [Low Path];
        7 -> 4 [Low Path];
        8 -> 7 [High Path];
        3 [label="x2"];
        3 -> 0 [High Path];
        8 -> 3 [Low Path];
        0 [Terminal Box];
        1 [Terminal Box];
}

Time taken for sifting operation = 37.398961 ms
Done.
Press any key to continue . . .
```

Fig: result graph of sifting algorithm on node12.blif and operation time

Problem 4: CUDD implementation on apply algorithm

```
lizemings-MacBook-Pro:nanotrav Zeming$ ./nanotrav a.blif -second b.blif -apply xnor
************************************
ECE1733 Assign2 Problem 4
Zeming Li and Yuchen Wang

************************************
# ./nanotrav a.blif -second b.blif -apply xnor
# CUDD Version 3.0.0
==========================
Read the first network...
==========================
==========================
Read the second network...
==========================
==========================
Initialize Ddmanager...
==========================
==========================
implement APPLY...
==========================
.model first
.inputs a b c d
.outputs A
.names a b c d A
--00 1
110- 1
1-11 1
10-0 1
.end
==========================
.model second
.inputs a b c
.outputs B
.names a b c B
111 1
-00 1
000 1
0-- 1
.end
==========================
Ddmanager has 10 nodes...
```

Fig: input data in two blif files.t

```
Network 1 has 4 inputs, 1 outputs
inputs:
a       b       c       d
outputs:
A
Node of network1 index=0 0x0
Node of network1 index=1 0x1
Node of network1 index=2 0x2
Node of network1 index=3 0x3
Node of network1 index=0 0x0
there are 5 nodes in Network 1
Network 2 has 3 inputs, 1 outputs
inputs:
a       b       c
outputs:
B
Node of network2 index=0 0x0
Node of network2 index=1 0x1
Node of network2 index=2 0x2
Node of network2 index=0 0x0
there are 4 nodes in Network 2
input 0  of Network 1 has var index 0
input 0 of Network 1 has support size: 1
input 0  of Network 2 has var index 0
input 0 of Network 2 has support size: 1
f is
: 7 nodes 1 leaves 64 minterms
ID =  0x3fedc58024b     index = 0       T = 0x3fedc58024a       E = !0x3fedc580239
ID =  0x3fedc580239     index = 2       T = 1           E =  0x3fedc580238
ID =  0x3fedc580238     index = 3       T = 1           E = !1
ID =  0x3fedc58024a     index = 1       T = 0x3fedc580242       E =  0x3fedc580249
ID =  0x3fedc580249     index = 2       T = 1           E = !0x3fedc580238
ID =  0x3fedc580242     index = 2       T = 0x3fedc580238       E =  1

g is
: 4 nodes 1 leaves 12 minterms
ID =  0x3fedc580253     index = 0       T = 0x3fedc58024f       E =  1
ID =  0x3fedc58024f     index = 1       T = 0x3fedc580237       E = !0x3fedc580237
ID =  0x3fedc580237     index = 2       T = 1           E = !1

h is
: 1 nodes 1 leaves 2 minterms
ID =  0x3fedc580231     value = 1

one->index=2147483647, zero->index=310378495
Time taken for apply = 0.006000 ms
```

Fig: input table and xnor operation result table. Operation time is measured.

Discussion

The algorithm for this program follows the reference material posted on portal for constructing BDD and performing apply operation. The sifting algorithm changes variable order and re-build BDD to achieve minimum node count. A variable is moved up and down in the order so that it takes all possible positions. The best position is identified and the variable is returned to that position. All possible locations for individual variable are tested. The program will first perform forward swap on the variable of interest, count nodes of each BDD. After forward swap, the program will restore the original BDD diagram and perform a backward swap and count nodes of each BDD. In the end, the program will move the variable to the optimal location which produces the minimum BDD.

The apply algorithm is written in as a separate function for clarity. The boolean operator strings need to be translated to its numerical values. All nodes from two BDDs are merged in one table and a new BDD diagram will be built. The second function will count the node from end of the first BDD diagram table. For example the test case shows the table for g will start from node 10. This feature could be improved to show more clarity.

In part IV, we rewrote and used the nanotrav/main.c function in CUDD package to implement apply option. The read option in CUDD package is different from what we used for assignment 2. Hence we have to modify the data structure and add the apply algorithm function. There is no “apply” function for BDDs, but CUDD package has two functions for node operation. Hence it is sufficient to implement all two-argument Boolean functions with these functions and the complementations. We have achieved boolean operation for AND, OR, NAND, NOR, XNOR for two functions and recorded the operation time. The operation time is shorter than our program but not significantly shorter in the basic test cases.

Limitation

When there are don’t care cubes in the function, the program will compare it with the adjacent node and assign the same value of the adjacent node to the DC node. This way we can eliminate the DC node. There is a possible way to make use of the assignment 1 program to preprocess the function to find the prime Implicants such that the build process can be reduced.

Secondly, the sifting algorithm requires large amount of computing memory to store temporary heap table and this program has only tested on simple cases. When the cube nodes are significantly larger, we could set up a time out option and handle this situation.

We have added an option to output the result to a .dot file. The .dot file can be converted to .png file for a more intuitive BDD result. However, the .dot file can only be converted in linux environment and requires additional tool to achieve this conversion in windows.