

## ECE156B Project 2

### Programming Assignment

I re-wrote the podem parser in Python 3.4, and used the PyEDA BDD manipulation package instead of 'CUDD'. This was significantly easier than I thought it would be as I was able to re-write the entire software package in 2 files totaling a 121 lines and 21 lines respectively (input.py and gate.py).

### Required Software

1. Python 3.4 (<https://www.python.org/download/releases/3.4.0/>)
2. PyEda 0.26 (<http://pyeda.readthedocs.org/en/latest/>)
3. GraphViz
4. Erdos (Online GraphViz Viewer) <http://sandbox.kidstrythisathome.com/erdos/>  
(Optional)

### Sample Execution

The second argument in the command-line execution is the circuit (.sim) file that you would want to generate .dot BDD files for. One execution of this script will create up to a maximum of 5 output files, appending the name of the output gate to the input filename as the output. If the '-i' flag is included the program will output the number of minterms and nodes included.

With '-i'	Metehans-MacBook-Air:ECE156b metehan\$ python3.4 input.py C17.sim -i Minterms : 1 Nodes : 6 Writing: 22GAT10_C17.dot Minterms : 1 Nodes : 6 Writing: 23GAT9_C17.dot
Without '-i'	Metehans-MacBook-Air:ECE156b metehan\$ python3.4 input.py C17.sim Writing: 22GAT10_C17.dot Writing: 23GAT9_C17.dot Output files generated. Exiting

Note: Although my program outputs the combinational circuit output in ROBDD format, the ordering specified from within the report was not used because I could not successfully implement the functionality and ran out of time. Although the BDDs are reduced and ordered, they are not ordered in the ordering pairs specified in the project guidelines.

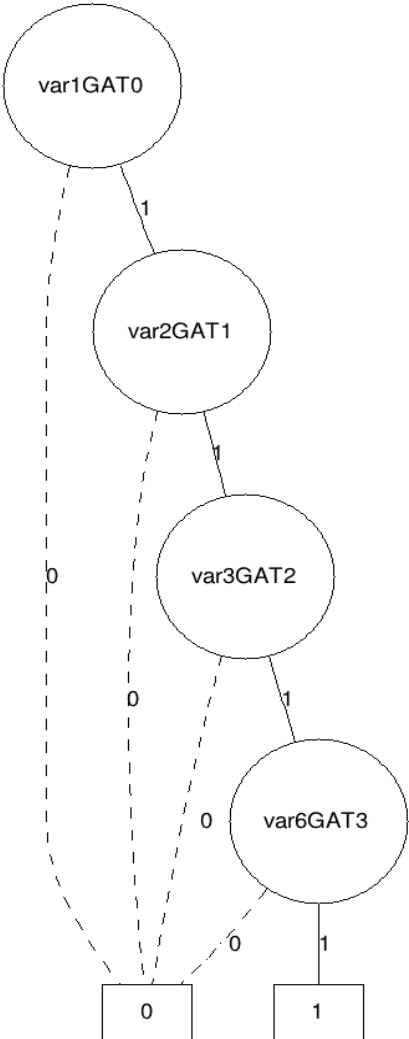
### How the BDDs were Built

I parsed the .sim circuit files in the input.py file, specifically within the parse\_circuit() function. After the circuits were parsed into the necessary global variables, I called a recursive

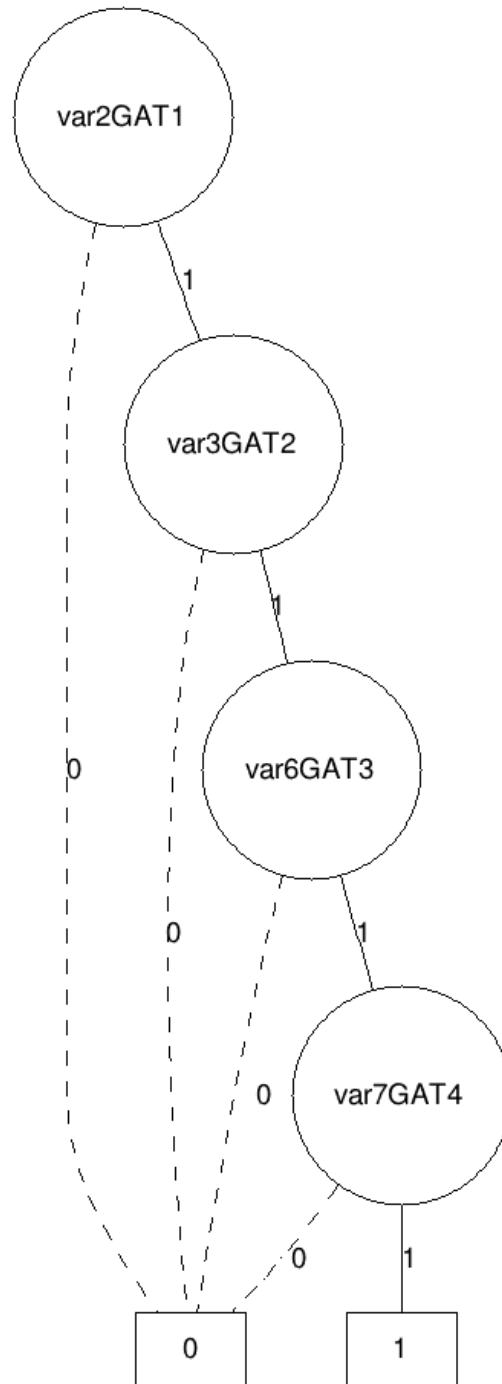
function called `build_expression(string)` on the primary outputs. This function call recursively assembled a string representation of the boolean expression represented by the circuit. After that, I simply called `expr2bdd()` on the string returned from `build_expression()` and outputted the necessary '.dot' files.

## Results

### C17.sim

Filename	Image	Information
22GAT10_C17.dot		Minterms: 1 Nodes: 6

23GAT9\_C17.dot



Minterms: 1  
Nodes :6

### C18.sim

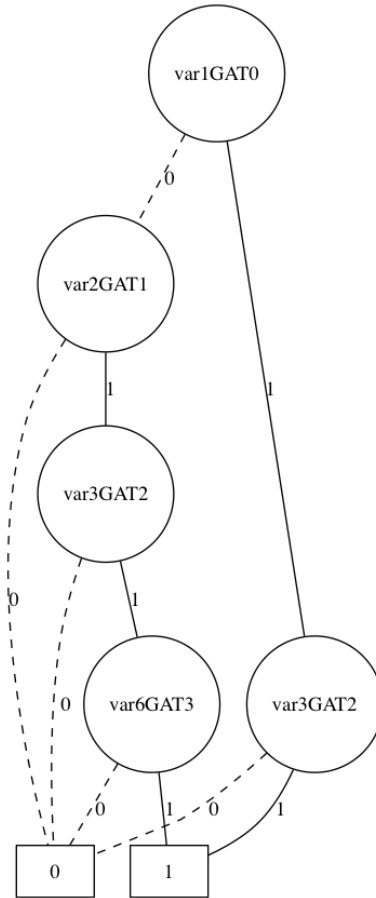
(Custom Circuit made by me for testing sake, similar to C17.sim except includes 'not' and 'or', for the sake of testing those gates)

Script Call	Metehans-MacBook-Air:ECE156b metehan\$ python3.4 input.py C18.sim -i
Script Input	Metehans-MacBook-Air:ECE156b metehan\$ python3.4 input.py C18.sim -i Minterms : 2 Nodes : 7 Writing: 29GAT67_C18.dot Minterms : 3 Nodes : 5 Writing: 30GAT68_C18.dot

C18.sim file contents
<pre>name C18.iscas i 1GAT(0) i 2GAT(1) i 3GAT(2) i 6GAT(3) i 7GAT(4)  o 29GAT(67) o 30GAT(68)  g1 and 6GAT(3) 3GAT(2) ; 11GAT(5) g2 and 3GAT(2) 1GAT(0) ; 10GAT(6) g3 and 7GAT(4) 11GAT(5) ; 19GAT(7) g4 and 11GAT(5) 2GAT(1) ; 16GAT(8) g5 and 19GAT(7) 16GAT(8) ; 23GAT(9) g6 and 16GAT(8) 10GAT(6) ; 22GAT(10) g7 or 16GAT(8) 10GAT(6) ; 29GAT(67) g8 not 16GAT(8) ; 30GAT(68)</pre>

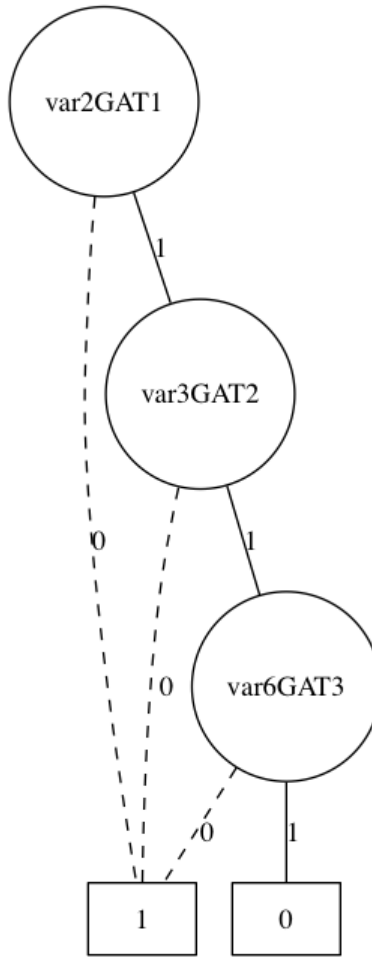
Filename	Image of BDD	Information
----------	--------------	-------------

29GAT67\_C18.dot



Minterms : 2  
Nodes : 7

30GAT68\_C18.dot

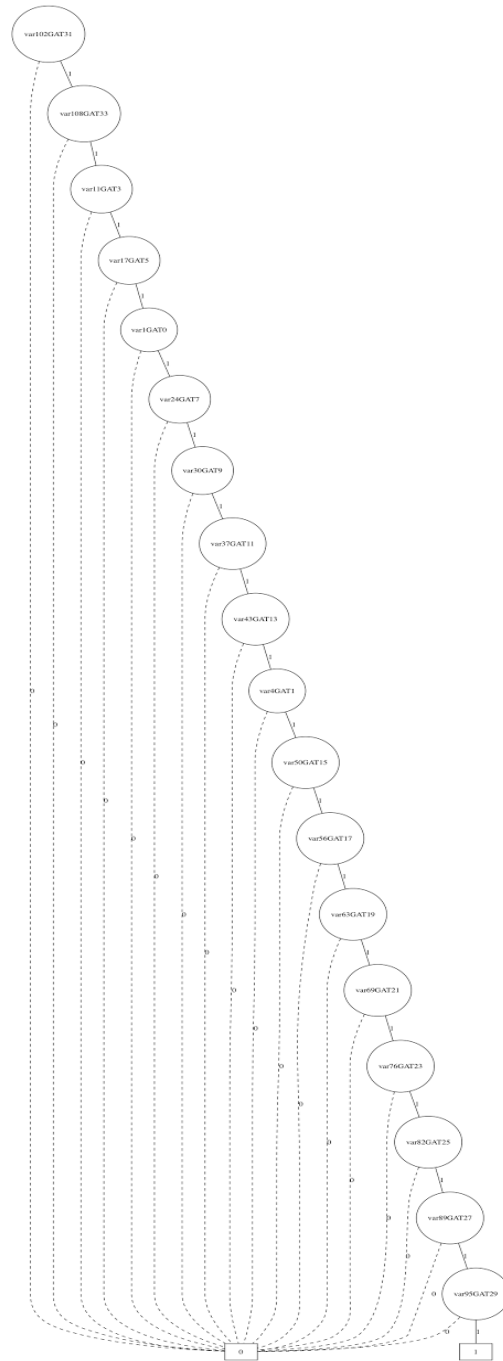


Minterms : 3  
Nodes : 5

**C432.sim**

Filename	Image of BDD	Information
----------	--------------	-------------

223GAT84\_C432.sim



Minterms : 1  
Nodes : 20

Only the first input was included for this larger circuit. If more inputs are desired simply run the code provided with the attachment this report was included in.

## Conclusion

Overall, developing this project using Python 3.4 (and PyEDA) was significantly easier than using the provided C packages. I was able to re-write the entire parser including with the BDD converter in under ~160 lines of python code.