ECE 5534 SPRING 2020 LAB #1 DUE: 2/17/2019

THIS LAB IS TO BE DONE INDIVIDUALLY.

Part 1: Fun with ROBDDs

In a high-level language of your choosing, implement the `Mk`, `Build`, `Apply`, `Restrict`, `SatCount` and `AnySat` functions specified in H.K. Andersen's *An Introduction to Binary Decision Diagrams* paper available on Canvas. You are not required to use dynamic programming optimization in your code.

Part 2: Demonstrating Your Code

You have two choices for evaluating Boolean expressions:

1. [*Basic*] In the implementation language of your choice, define:
   a. The Boolean operations as functions (`not, and, or, imp, equiv`) with one or two arguments.
   b. `x[i]` as a Boolean array of `0/1` values. When you want to set or change the value of a variable just set array element `x[i]` to either `0` or `1`.
   c. The overall Boolean expressions as functions expressed in terms of the primitive Boolean functions defined in (a), variables `x[i]`, and Boolean constants `0` and `1` as in the above example.

2. [*If you have some experience with parsing*] Define and parse (from standard input) a prefix notation for Boolean logic. A possible format is the following:

Syntax	Semantics
`not(expr)`	negation
`and(lexpr, rexpr)`	conjunction
`or(lexpr, rexpr)`	disjunction
`imp(lexpr, rexpr)`	implication
`equiv(lexpr, rexpr)`	equivalence

   The constants are `0` and `1`. All variables are of the form `x1`, `x2`, `x3`, … A sample formula is "`and(imp(not(x1),equiv(1,x2)),not(x2))`". White space should be ignored. Write a simple lexical analyzer to recognize the above tokens (Boolean operators, variables, constants, parentheses, commas). The lexical analyzer can be called by a recursive descent parser to create a syntax tree for the expression. Your ROBDD

code can then recursively walk the syntax tree to evaluate the expression. You may assume that all input expressions are legal.

After initial debugging with examples from the Andersen paper, either manually or automatically generate non-trivial, random expressions for testing the functions you have developed. Points will be awarded for the comprehensiveness of your testing, including examples that require considerable time and/or memory. Consider using ROBDDs to show the equivalence or non-equivalence of two Boolean expressions, such as unoptimized and optimized variants of the same function.

Part 3: What to Submit

Create a directory containing **only** your source code and a **PDF** document demonstrating the correct operation of your code. The grading sheet should serve as a guide for what to include in your test suite and report. A ZIP archive of the directory should be submitted to Canvas before midnight on Monday February 17. The archive should be named `Lab1_your_pid.zip`, where `your_pid` is replaced with your PID.