Logic Synthesis & Verification, Fall 2025

National Taiwan University

Programming Assignment 1

Exercises 1-3 are due on 9/21 23:59. Exercise 4 is due on 10/5 23:59.
Please submit Exercises 1 and 4 via GitHub, and Exercises 2 and 3 via NTU Cool.

Submission Guidelines. For Exercises 2 and 3, please compress the required item into a single `.tgz` file and submit it via NTU cool. For Exercise 4, please develop your code under `src/ext-lsv`. You are asked to submit your assignments by creating pull requests to your own branch. To avoid plagiarism, please create pull requests between 21:00 and 23:59 on the due date. Please see the GitHub page (https://github.com/NTU-ALComLab/LSV-PA) for more details.

1 [Getting Familiar with GitHub] (0%)

(a) Open the GitHub page and check whether there is your own branch named by your student ID number. If you cannot find your own branch or your branch is inconsistent with the `master` branch, please contact TA.
(b) Fork the repository to your personal GitHub account. You will need to develop your programs on your forked repository.
(c) Edit `participants-id.csv` under `lsv/admin/` to register your name and GitHub account (next to your student ID). Send a pull request to the `master` branch after you finish it. Note that this is the only time you send a pull request to the `master` branch. For Exercise 4 and future programming assigments, you are asked to send pull requests to your own branch.

2 [Using ABC] (10%)

(a) Create a BLIF file named "`alu.blif`" to represent a 2-bit ALU (arithmetic logic unit) with 5 inputs a_1, a_0, b_1, b_0, c, and 3 outputs y_2, y_1, y_0.
The inputs $A = (a_1, a_0)$ and $B = (b_1, b_0)$ are the binary encoding of two unsigned 2-bit integers (with a_0, b_0 being the less significant bit,) and c is to control whether to perform addition or subtraction.
The output $Y = (y_2, y_1, y_0)$ is a 3-bit signed integer in 2's complement (with y_0 being the less significant bit). When $c = 0$, the circuit performs addition and the output $Y = A + B$; on the contrary, when $c = 1$, the circuit performs subtraction and the output $Y = A - B$.
For example, when inputs $(a_1, a_0, b_1, b_0, c) = 10110$, the circuit should perform $2 + 3 = 5$, and the output $(y_2, y_1, y_0) = 101$ (please ignore the overflow issue).
You can find the BLIF manual in `http://www.eecs.berkeley.edu/~alanmi/publications/other/blif.pdf`. For reference, there are also some BLIF files in the directory "`lsv/pa1/benchmark`".

(b) Perform the following steps to practice using ABC with your "`alu.blif`". Screenshot the results after running the commands and put them in your report.
   1. read the BLIF file into ABC (command "`read`")
   2. check statistics (command "`print_stats`")
   3. visualize the network structure (command "`show`")
   4. convert to AIG (command "`strash`")
   5. visualize the AIG (command "`show`")
   6. convert to BDD (command "`collapse`")
   7. visualize the BDD (command "`show_bdd -g`"; note that "`show_bdd`" only shows the first PO; option "`-g`" can be applied to show all POs)

3 [ABC Boolean Function Representations] (10%)

In ABC, there are different ways to represent Boolean functions.

(a) Compare the following differences with your "`alu.blif`". Screenshot the results and briefly describe your findings in your report.
   1. logic network in AIG (by command "`aig`") vs.
      structurally hashed AIG (by command "`strash`")
   2. logic network in BDD (by command "`bdd`") vs.
      collapsed BDD (by command "`collapse`")

(b) Given a structurally hashed AIG, find a sequence of ABC commands to convert it to a logic network with node function expressed in sum-of-products (SOP). Use your "`alu.blif`" to test your command sequence (by first running "`strash`" to convert it to AIG). Screenshot the results, and put them in your report.

4 [Programming ABC] (80%)

In this problem, you are asked to write your own procedures and integrate them into ABC, so that your self-defined commands can be executed within ABC. You may need to trace some source code in ABC to understand the data structures and function usages.

Hint 1: You may refer to `src/base/abc/abc.h` to find definitions of some important variables, functions, iterators, etc.

Hint 2: You may use the command "`grep -R <keyword>`" or your code editor to find whether a certain keyword appears in the source code. It can be helpful to see how these functions or data structures are used in the source code.

Hint 3: More tips, FAQs, and future updates will be posted on the GitHub wiki page (https://github.com/NTU-ALComLab/LSV-PA/wiki/PA1-2025)

4.1 Multi-output Cut Enumeration

A *k-feasible cut* of a node n is an irredundant cut with no more than k nodes. In other words, a cut is a k-feasible cut of node n if and only if no subset of this cut is a cut of node n, and the number of nodes in this cut is not more than k. A *k-l multi-output cut* is a k-feasible cut for at least l output nodes.

In this exercise, write a procedure in the ABC environment to enumerate k-l multi-output cuts in an AIG. Please print the inputs and outputs of every multi-output cut, considering all primary inputs and AND nodes. For simplicity, you may ignore the primary outputs. Integrate this procedure into ABC (under `src/ext-lsv/`) so that after reading in a circuit (by command "`read <filename>`") and transforming it into AIG (by command "`strash`"), running the command "`lsv_printmocut`" would invoke your code and prints the result. Your code will be tested using the designs in "`lsv/pa1/benchmarks`."

The command should have the following format.

```
lsv_printmocut <k> <l>
```

for $3 \leq k \leq 6$ and $1 \leq l \leq 4$. The output should print the cuts line by line (in arbitrary order). Each line is a multi-output cut in the following format:

```
<in_1> <in_2> ... : <out_1> <out_2> ...
```

where `<in_1>`, `<in_2>`, ... are the node IDs of a k-feasible cut, sorted ascendingly, and `<out_1>`, `<out_2>`, ... are the node IDs of all the output nodes that share this k-feasible cut, sorted ascendingly. IDs are separated by spaces, and the inputs and outputs are separated by a "`:`" character (with spaces around it).

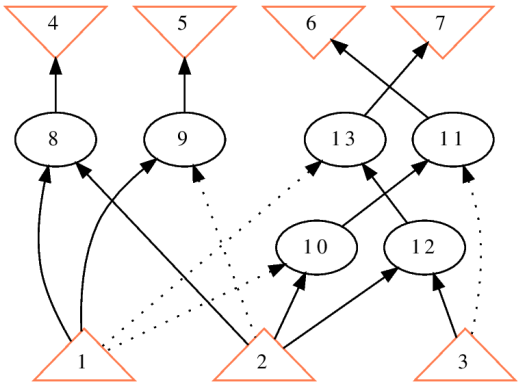

Fig. 1. A simple AIG

For example, given the AIG in Fig. 1, a command to compute 3-2 multi-output cuts and the corresponding output should look like:

```
abc 01> lsv_printmocut 3 2
1 2 : 8 9 10
1 2 3 : 11 13
```

Note that the primary outputs (4, 5, 6, and 7) are ignored, although they do share a unit cut with their input AND nodes.

Finally, there may be some built-in functions in ABC for cut enumeration. You are allowed to refer to them, but you have to write your own procedures. Directly calling or copying from the built-in functions (of cut enumeration) will be viewed as plagiarism.

Files to Submit

1. Modify "`participants-id.csv`" (by pull request to branch "`master`").
2. The BLIF file in exercise 2(a).
3. The PDF report named `report.pdf`.
4. The source code in problem 4 (by pull request to your own branch named by your student ID).