Logic Synthesis & Verification, Fall 2024

National Taiwan University

Programming Assignment 1

Exercises 1-3 are due on 9/22 23:59; Exercise 4 is due on 10/6 23:59.
Please submit Exercises 1 and 4 on GitHub, and Exercises 2 and 3 on NTU Cool.

Submission Guidelines. For Exercises 2 and 3, please put the required items under `lsv/pa1/`, i.e., this folder. Compress the `lsv/pa1/` folder as a single `.tgz` file and submit it on 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 push files and 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 student ID, name, and GitHub account. 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. In the following, you are asked to send pull requests to your own branch.

2 [Using ABC] (10%)

(a) Create a BLIF file named "`comp.blif`" to represent a 5-to-3 compressor with 5 inputs x_0, x_1, x_2, x_3, and x_4. Its output $Y = (y_2, y_1, y_0)$ is a 3-bit unsigned integer that represents the number of 1's in the inputs. For example, when inputs are 10011, output Y = $(011)_2$, i.e., binary number of 3, because there are three 1's in 10011. You can find the BLIF manual in `http://www.eecs.berkeley.edu/~alanmi/publications/other/blif.pdf`.
(b) Perform the following steps to practice using ABC with your "`comp.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 the two-bit unsigned multiplier example. 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 "`comp.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 codes 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 editer to find whether a certain keyword appears in the source code. It can be very helpful seeing how these functions or data structure are used in the source code.

4.1 *k*-feasible Cut Enumeration

Write a procedure in the ABC environment to enumerate all k-feasible cuts of every node on an AIG. Integrate this procedure into ABC (under `src/ext-lsv/`) so that after reading in a circuit (by command "`read`") and transforming it into AIG (by command "`strash`"), running the command "`lsv_printcut`" would invoke your code and prints the result.

The command should have the following format.

```
lsv_printcut <k>
```

Where the parameter <k> specifies k-feasible cuts, for $3 \leq k \leq 6$. The output should have the following format

```
<node_1>: <cut_1>
<node_1>: <cut_2>
        ...
<node_2>: <cut_1>
        ...
```

where <node_i> is a node ID, and each <cut_j> after <node_i> is a k-feasible cut for <node_i>. <cut_j> should be a sequence of node IDs, sorted ascendently, separated by space characters.

```
<node_1> <node_2> <node_3> ...
```

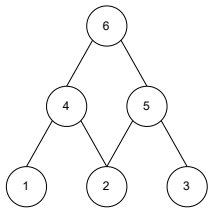

Fig. 1. A simple AIG

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

```
abc 01> lsv_printcut 3
1: 1
2: 2
3: 3
4: 4
4: 1 2
5: 5
5: 2 3
6: 6
6: 4 5
6: 1 2 5
6: 2 3 4
6: 1 2 3
```

Note that there may be some built-in functions in ABC computing 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 will be viewed as plagiarism.

Files to Submit

1. The BLIF file in problem 2(a).
2. The PDF report named `report.pdf`.
3. The source code in problem 4 (by pull request).