Logic Synthesis & Verification, Fall 2023

National Taiwan University

Programming Assignment 2

Due on 11/30 23:59 on GitHub.

Submission Guidelines. Please develop your code under `src/ext-lsv`. Please do not modify any code outside `src/ext-lsv`, and we will only copy your files under `src/ext-lsv` for evaluation. 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 at the last moment before the deadline. Please see the GitHub page (https://github.com/NTU-ALComLab/LSV-PA) for more details.

1 [Symmetry Checking with BDD] (50%)

Given a circuit C in BDD form, an output pin y_k, and two input variables x_i, x_j, write a procedure in ABC to check whether the output pin y_k is symmetric in x_i and x_j. If not, show a counterexample to prove it. Integrate this procedure into ABC (under `src/ext-lsv/`), so that after reading in a circuit (by command "`read`") and transforming it into BDD (by command "`collapse`"), running the command "`lsv_sym_bdd`" would invoke your code. The command should have the following format.

`lsv_sym_bdd <`k`> <`i`> <`j`>`

where k is the output pin index starting from 0, and i and j are input variable indexes starting from 0. If the k^{th} output is symmetric in the i^{th} and the j^{th} variable, just print "`symmetric`", as shown below.

```
symmetric
```

Otherwise, print "`asymmetric`" and show a counterexample in the following format.

```
asymmetric
<pattern 1>
<pattern 2>
```

The values in the input pattern follow the input variable order. Note that your counterexample should be able to prove the asymmetry. In other words, patterns 1 and 2 should (1) lead to different output values for the k^{th} output pin and (2) only differ in exchanging the values of the i^{th} and the j^{th} variable.

For example, suppose the BDD has only one output pin representing the function $y_0 = x_0x_1 + x_2$. The command and the output should look like:

```
abc 01> lsv_sym_bdd 0 0 1
symmetric
abc 02> lsv_sym_bdd 0 0 2
asymmetric
100
001
```

where the counterexample shows that $(x_0, x_1, x_2) = (1, 0, 0)$ and $(x_0, x_1, x_2) = (0, 0, 1)$ will lead to different y_0 values.

Notice. When operating BDDs, remember to use `Cudd_Ref` when you create a BDD node and use `Cudd_RecursiveDeref` when you dereference a BDD node. This helps to avoid `cuddGarbageCollect` errors. Here is an example showing how to use these commands.

```
Ddnode* cube = Cudd_ReadOne(manager);
for (int i = 0; i < n; ++i) {
    Ddnode* var = Cudd_bddIthVar(manager, i);
    Cudd_Ref(var);
    Ddnode* new_cube = Cudd_bddAnd(manager, cube, var);
    Cudd_Ref(new_cube);
    Cudd_RecursiveDeref(manager, cube);
    Cudd_RecursiveDeref(manager, var);
    cube = new_cube;
}
```

2 [Symmetry Checking with SAT] (50%)

Repeat Exercise 1 with all conditions being the same except that the circuit C is in the form of AIG (by commands "`read`" and "`strash`"). Use SAT solver to check whether the output pin y_k is symmetric in x_i and x_j. Your procedure should implement the command in the following format.

`lsv_sym_sat <`k`> <`i`> <`j`>`

where k is the output pin index starting from 0, and i and j are input variable indexes starting from 0.

The output format is the same as in Exercise 1. For example, suppose the logic network has only one output pin, which represents the function $y_0 = x_0x_1 + x_2$. The command and the output should look like:

```
abc 01> lsv_sym_sat 0 0 1
symmetric
abc 02> lsv_sym_sat 0 0 2
asymmetric
100
001
```

Hint 1. You can follow the following steps.

(1) Use `Abc_NtkCreateCone` to extract the cone of y_k.
(2) Use `Abc_NtkToDar` to derive a corresponding AIG circuit.
(3) Use `sat_solver_new` to initialize an SAT solver.
(4) Use `Cnf_Derive` to obtain the corresponding CNF formula C_A, which depends on variables $v_1, \ldots, v_n$.
(5) Use `Cnf_DataWriteIntoSolverInt` to add the CNF into the SAT solver.
(6) Use `Cnf_DataLift` to create another CNF formula C_B that depends on different input variables $v_{n+1}, \ldots, v_{2n}$. Again, add the CNF into the SAT solver.
(7) For each input x_t of the circuit, find its corresponding CNF variables $v_A(t)$ in C_A and $v_B(t)$ in C_B. Set $v_A(t) = v_B(t)\, \forall t \notin \{i, j\}$, and set $v_A(i) = v_B(j)$, $v_A(j) = v_B(i)$. This step can be done by adding corresponding clauses to the SAT solver.
(8) Use `sat_solver_solve` to solve the SAT problem. Note that y_k is symmetric in x_i and x_j if and only if $v_A(y_k) \oplus v_B(y_k)$ is unsatisfiable, where $v_A(y_k)$ and $v_B(y_k)$ are the CNF variables in C_A and C_B that corresponds to y_k.
(9) If y_k is asymmetric in x_i and x_j, use `sat_solver_var_value` to obtain the satisfying assignment, which can be used to derive the counterexample.

Hint 2. To use `Abc_NtkToDar` and `Cnf_Derive` functions, you should include the following code.

```
#include "sat/cnf/cnf.h"
extern "C"{
    Aig_Man_t* Abc_NtkToDar( Abc_Ntk_t * pNtk, int fExors, int fRegisters );
}
```

Hint 3. The variable orders in CNF differ from the ones in AIG. For a pointer `pObj` in `Abc_Obj_t*` type, you can use `pCnf->pVarNums[pObj->ID]` to find its variable index in `pCnf`.

Hint 4. You can refer to our GitHub page (https://github.com/NTU-ALComLab/LSV-PA/wiki/Reasoning-with-SAT-solvers) for more details about using SAT solvers in ABC.