Homework # 5 (version 1)

Due Wed. May 14, 2016

1. **Synthesis of Cyclic Combinational Circuits**

   The goal in multilevel logic synthesis is to obtain the best multilevel, structured representation of a network. The process typically consists of an iterative application of minimization, decomposition, and restructuring operations. An important operation is **substitution**, in which node functions are expressed, or re-expressed, in terms of other node functions as well as of their original inputs. Consider the target functions in Figure 1.

$$\begin{aligned} f_1 &= \bar{x}_1 x_2 \bar{x}_3 + \bar{x}_2(x_1 + x_3) \\ f_2 &= \bar{x}_1 \bar{x}_2 \bar{x}_3 + x_1(x_2 + x_3) \\ f_3 &= \bar{x}_3(\bar{x}_1 + \bar{x}_2) + \bar{x}_1 \bar{x}_2 \end{aligned}$$

Figure 1: Target functions for synthesis.

Substituting f_3 into f_1, we get

$$f_1 = f_3(x_1 + x_2) + \bar{x}_2 x_3.$$

Substituting f_3 into f_2, we get

$$f_2 = \bar{x}_1 \bar{x}_2 \bar{x}_3 + x_1 \bar{f}_3.$$

Substituting f_2 and f_3 into f_1, we get

$$f_1 = \bar{x}_2 x_3 + \bar{f}_2 f_3.$$

For each target function, we can try substituting different sets of functions. Call such a set a **substitutional set**. Different substitutional sets yield alternative

functions of varying cost. In general, augmenting the set of functions available for substitution leaves the cost of the resulting expression unchanged or lowers it. (Strictly speaking, this may not always be the case since the algorithms used are heuristic.)

In existing methodologies, a total ordering is enforced among the functions in the substitution phase to ensure that no cycles occur. This choice can influence the cost of the solution. Consider the ordering:

$$\begin{aligned} f_1 &= \bar{x}_2 x_3 + \bar{f}_2 f_3 \\ f_2 &= \bar{x}_1 \bar{x}_2 \bar{x}_3 + x_1 \bar{f}_3 \\ f_3 &= \bar{x}_3(\bar{x}_1 + \bar{x}_2) + \bar{x}_1 \bar{x}_2. \end{aligned}$$

This has a cost of 14. (As discussed in class, our **cost measure** for area is the number of literals in node expressions.)

Enforcing an ordering is limiting since functions near the top cannot be expressed in terms of very many others (the one at the very top cannot be expressed in terms of *any* others). Dropping this restriction can lower the cost. For instance, if we allow every function to be substituted into every other, we obtain:

$$\begin{aligned} f_1 &= \bar{x}_2 x_3 + \bar{f}_2 f_3 \\ f_2 &= x_1 \bar{f}_3 + \bar{x}_3 \bar{f}_1 \\ f_3 &= f_1 \bar{f}_2 + \bar{x}_2 \bar{x}_3 \end{aligned}$$

This has cost of 12. This network is cyclic and it is *not* combinational. As was discussed in class, an essential step in the synthesis process is the verify whether a circuit is combinational.

A cyclic solution that *is* combinational is:

$$\begin{aligned} f_1 &= \bar{x}_3 \bar{f}_2 + \bar{x}_2 x_3 \\ f_2 &= \bar{x}_1 \bar{x}_2 \bar{x}_3 + x_1 \bar{f}_3 \\ f_3 &= \bar{x}_1 f_1 + \bar{x}_2 \bar{x}_3 \end{aligned}$$

This has cost 13. So, introducing cycles can help reduce the cost, and yet still produce a valid solution.

Problem

Consider the functions:

$$\begin{aligned} f_1 &= ab\bar{c} + \bar{b}c \\ f_2 &= \bar{a}\bar{b}c \\ f_3 &= \bar{c}(\bar{b} + \bar{a}) + \bar{a}\bar{b} \end{aligned}$$

Design a combinational network, possibly cyclic, with minimal cost (where the cost is measured as the literal count.) For full points, find a network with cost 10.

2. **Implementing XOR with AND gates**

   Recall that the Exclusive-OR (XOR) function is 1 if an odd number of the inputs are 1, and 0 otherwise. Suppose that we have to build an circuit that computes XOR with AND gates, OR gates, and inverters.

   (a) **Trees**

      i. Draw a circuit to compute the XOR of 13 variables with the fewest possible number of AND and OR gates gates, and as many inverters as you want. (Solutions has 30 AND and OR gates.)

      ii. Draw a circuit to compute the XOR of 15 variables with the fewest possible number of AND and OR gates gates, and as many inverters as you want. (Solutions has 35 AND and OR gates.)

      iii. Draw a circuit to compute the XOR of 23 variables with the fewest possible number of AND and OR gates gates, and as many inverters as you want. (Solutions has 55 AND and OR gates.)

      iv. Draw a circuit to compute the XOR of 27 variables with the fewest possible number of AND and OR gates gates, and as many inverters as you want. (Solutions has 65 AND and OR gates.)

   (b) **A Lower Bound**

   Solve any one of the following for full points.

      i. Write a computer program that verifies that no circuit with 7 AND/OR gates computes the XOR of 4 variables. (You can limit the number of configurations that it tries up to symmetries, e.g., permuting the variables. But you have to convince me that you have verified all distinct configurations, up to such symmetries.)

      ii. Write a program that verifies that no circuit with x AND/OR gates computes the XOR up to y variable. (**Bonus points if you can go above $x = 7$, $y = 4$.**)

      iii. Improve the lower bound to $\lceil 2.5n - 1 \rceil$. (I don't know if this is true. In addition to full points for this problem, **I'll give you an immediate A in the course if you solve this**.)

      iv. Discuss the implementation of the XOR of n variables in terms of CMOS transistors. What is the minimum number of transistors that you require? Can you prove a lower bound (or prove that your construct is optimal)?

3. **Quantification**

   In this problem, you will implement so-called **quantification** operations on Boolean functions.

   We use the standard notation: addition (+) denotes disjunction (OR), multiplication ($\cdot$) denotes conjunction (AND), and an overbar ($\bar{x}$) denotes negation (NOT). The **restriction** operation (also known as the cofactor) of a function f with respect to a variable x, denoted as

   $$f|_{x=v},$$

   refers to the assignment of the constant value $v \in \{0, 1\}$ to x. A function f **depends** upon a variable x iff $f|_{x=0}$ is not identically equal to $f|_{x=1}$. Call the variables that a function depends upon its **support set**.

   A quantification operation on a function f is specified with respect to a subset of the variables in its support set; we will call this the "quantization list" and we'll use $y_1, \ldots, y_n$ for these. The quantization operation yields a new function, g, with a support set that consists of only of variables in the support set of f than were *not* in the quantization list. We'll use $x_1, \ldots, x_m$ for these.

   So, for given a function $f(x_1, \ldots, x_m, y_1, \ldots, y_n)$:

   - the **universal quantification** operation (also known as consensus) yields a function
     $$g(x_1, \ldots, x_m) = \forall (y_1, \ldots, y_n) f$$
     that equals 1 for assignments to $x_1, \ldots x_m$ where f equals 1 for all 2^n assignments to $y_1, \ldots, y_n$.
   - The **existential quantification** operation (also known as smoothing) yields a function
     $$g(x_1, \ldots, x_m) = \exists (y_1, \ldots, y_n) f$$
     that equals 1 for assignments to $x_1, \ldots x_m$ where f equals 1 for any of the 2^n assignments to $y_1, \ldots, y_n$.
   - Let's define the **"down" operation** as follows. It yields a function
     $$g(x_1, \ldots, x_m) = f \downarrow (y_1, \ldots, y_n)$$
     that equals 1 for assignments to $x_1, \ldots x_m$ where, for each variable y_i in the quantization list, there is some assignment to the remaining variables $y_j, j \neq i$, such that f depends on y_i. (This operation is commonly known as the *Boolean difference*.)

- Let's define the **"up" operation** as follows. It yields a function

$$g(x_1, \ldots, x_m) = f \uparrow (y_1, \ldots, y_n)$$

that equals 1 for assignments to $x_1, \ldots x_m$ where f is invariant (either identically 0 or identically 1) for all 2^n assignments to $y_1, \ldots, y_n$. (For a single variable, the "up" operation is the same as the complement of the "down" operation. However, for more than one variable, it is *not* the same.)

Examples

For

$$f(x_1, x_2, x_3) = x_1(x_2 + x_3),$$

we have,

- $\forall x_2 \;\; f = x_1 x_3$
- $\exists x_3 \;\; f = x_1$
- $f \downarrow x_1 = x_2 + x_3$
- $f \uparrow x_1 = \bar{x}_2 \bar{x}_3$

For

$$f(x_1, x_2, x_3, x_4, y_1, y_2) = x_1 + x_2 y_1 + x_3 y_2 + x_4 y_1 y_2,$$

we have

$$\begin{aligned} f \downarrow y_1 &= \bar{x}_1(\bar{x}_3(y_2 x_4 + x_2) + x_2 \bar{y}_2) \\ f \downarrow y_2 &= \bar{x}_1(\bar{x}_2 y_1 x_4 + x_3(\bar{y}_1 + \bar{x}_2)) \\ f \downarrow (y_1, y_2) &= \bar{x}_1(\bar{x}_2 \bar{x}_3 x_4 + x_2 x_3) \\ f \uparrow y_1 &= x_1 + x_3 y_2 + \bar{x}_2(\bar{x}_4 + \bar{y}_2), \\ f \uparrow y_2 &= x_1 + x_2 y_1 + \bar{x}_3(\bar{x}_4 + \bar{y}_1), \\ f \uparrow (y_1, y_2) &= x_1 + \bar{x}_2 \bar{x}_3 \bar{x}_4. \end{aligned}$$

Problem Specification

You have two options: either (a) answer questions and perform calculations of $\forall, \exists, \downarrow$, and $\uparrow$ manually; or (b) write some code.

(a) **Questions and Calculations**

i. Consider how you can compute these for operations symbolically, that is to say, with Boolean operations.

A. How can you compute the $\forall$ operation with the restriction operation and standard Boolean operations?

B. How can you compute the $\exists$ operation with the restriction operation and standard Boolean operations?

C. How can you compute the $\uparrow$ operation with the restriction operation and standard Boolean operations?

D. How can you compute the $\downarrow$ operation with the restriction operation and standard Boolean operations?

ii. Given an example of a function for which the "up" operation is not the same as the complement of the "down" operation.

iii. Compute $\forall(y_1, \ldots, y_n), \exists(y_1, \ldots, y_n), \downarrow(y_1, \ldots, y_n)$, and $\uparrow(y_1, \ldots, y_n)$ for the following functions.

A. $f(x_1, y_1, y_2, y_2, y_4) = x_1 y_1 y_2 y_3 y_4$

B. $f(x_1, y_1, y_2, y_2, y_4) = x_1 + y_1 + y_2 + y_3 + y_4$

C. $f(x_1, y_1, y_2, y_2, y_4) = x_1 \oplus y_1 \oplus y_2 \oplus y_3 \oplus y_4$

D. $f(x_1, x_2, x_3, y_1, y_2, y_3) = x_1(x_2 + x_3(y_1 + y_2 y_3))$

Here $+$ denotes OR; multiplication represents AND; $\oplus$ denotes Exclusive-OR; a over-bar represents NOT.

(b) **Coding**

Given a truth table representing a Boolean function and a quantization list, produce four truth truth tables corresponding to the application of these operations: universal quantification, existential quantification, "down" quantification, and "up" quantification.

Input Specification

The input consists of three non-negative integers.

- The first is the number of variables in the truth table.
- The second specifies the truth-table: the bits in the binary representation of the integer specify which rows of the truth table evaluate to 1. (The most significant bit is the last row of the truth table; the least significant bit is the first row.)
- The third specifies the quantization list. The bits in the binary representation of the integer specify whether the corresponding variable is included in the list (if the bit is 1) or not (if the bit is 0). The variables are ordered in the same way as for the columns of the truth table.

You can assume that you'll only get functions with five or fewer variables.

Output Specification

The output consists of four non-negative integers. These specify the truth tables for the result of universal quantification, existential quantification, "down" quantification, and "up quantification.

The number of variables in each of these truth tables is the original number minus the number in the quantization list. (The ordering of the variables is the same with those in the quantization list deleted.)

Again, the bits in the binary representation of the integer specify which rows of the truth table evaluate to 1. (The most significant bit is the last row of the truth table; the least significant bit is the first row.)

Example of Input and Output

For $f = x_1(x_2 + x_3)$, and a quantization list (x_2, x_3), the input is a text file

```
3
224
3
```

Note that $224_{10} = (11100000)_2$. This is the truth table for f. Note that $3_{10} = (011)_2$. This specifies (x_2, x_3) as the quantization list.

The output (written to `stdout`) is

```
0
2
2
1
```

These are the truth tables for the results of the four quantization operations. Each truth table is in terms of a single variable, x_1. Note that $0_{10} = (00)_2, 1_{10} = (01)_2$, and $2_{10} = (10)_3$.

Note that

$$\forall(x_2, x_3) \;\; f = 0$$

$$\exists(x_2, x_3) \;\; f = x_1$$

$$f \downarrow (x_2, x_3) = x_1$$

$$f \uparrow (x_2, x_3) = \bar{x}_1$$

The output specifies these truth tables in terms of x_1.