CS-472: Design Technologies for Integrated Systems

Programming Assignment 4 Due date: 24/11/2022, 24:00

1 Introduction

1.1 Binary Truth Tables

Truth tables are a compact way to represent single-output Boolean functions in computer programs. A 64 bit (unsigned) integer can be used to represent a Boolean function of at most 6 variables, where each bit correspond to the function output under a certain input assignment. Conventionally, bits are used from the least significant bit (LSB). For example, $f(x_2, x_1, x_0) = x_2 x_1 + x_1' x_0'$ can be represented with a binary number 11010001.

Table 1: Truth table of f.

x_2	x_1	x_0	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

1.2 Bit-Wise Operations

One of the advantages of representing Boolean functions with binary truth tables is fast logic computation. In many programming languages, bit-wise operations can be done with integers, meaning that we can compute the result of, for example, ANDing two functions with one machine instruction.

In the provided code, we have implemented a `Truth_Table` class (in `truth_table.hpp`), and overloaded some operators (in `operators.hpp`) to perform NOT, OR, AND and XOR operations with `Truth_Table` objects. Compile with `make` and run `./hw4` (without any argument) to see an example of computing f', $f + g$, $f \cdot g$ and $f \oplus g$, where the truth table of $g(x_2, x_1, x_0) = x_1$ is 11001100.

1.3 Bit Masks

Bit masks are useful for dealing with truth tables. For example, to extract the bits (the values of f) where $x_2 = 1$, we can bit-wise AND the truth table of f with a mask 11110000. The resulting truth table, 11010000, is 0 at the bit positions where $x_2 = 0$, and is equal to the truth table of f at the bit positions where $x_2 = 1$.

2 Task

The `Truth_Table` data structure is implemented and some operators are provided. Please complete the implementations of the following functions (in `cofactors.hpp`). Some useful bit masks are provided in `utils.hpp`. You can look up how `length_mask` is used in the constructor of `Truth_Table` to learn how to use the masks.

- `Truth_Table Truth_Table::positive_cofactor( uint8_t const var ) const`
  Compute the cofactor with respect to variable x_{var}. Note that variable indices start from 0.
- `Truth_Table Truth_Table::negative_cofactor( uint8_t const var ) const`
  Compute the cofactor with respect to variable x'_{var}.
- `Truth_Table Truth_Table::derivative( uint8_t const var ) const`
  Compute the Boolean derivative (Boolean difference) with respect to variable x_{var}.
- `Truth_Table Truth_Table::consensus( uint8_t const var ) const`
  Compute the consensus with respect to variable x_{var}.
- `Truth_Table Truth_Table::smoothing( uint8_t const var ) const`
  Compute the smoothing with respect to variable x_{var}.

The definitions of these operators can be found in the textbook on pages 69-70.

Note: After these operations, the resulting function no longer depends on x_{var}. That is, the half of the truth table where $x_{var} = 1$ should be the same as the other half where $x_{var} = 0$. However, we do not change the length of the truth table, i.e., `num_var` stays the same. This is to avoid re-indexing variables.

3 Compiling and Testing

With a command line interface, type `make` to compile the code. An executable named `hw4` will be produced if compilation is successful.

Type `./hw4 <binary_truth_table>` to test the code. You can insert some underscores (_) in the truth table for better readability. Do not insert spaces or other special characters. For example, run `./hw4 1101_0001` to test with the example function f above.

Example output:

```
./hw4 11010001
positive cofactor w.r.t. variable 0: 11000000
negative cofactor w.r.t. variable 0: 11110011
      derivative w.r.t. variable 0: 00110011
       consensus w.r.t. variable 0: 11000000
       smoothing w.r.t. variable 0: 11110011

positive cofactor w.r.t. variable 1: 11110000
negative cofactor w.r.t. variable 1: 01010101
      derivative w.r.t. variable 1: 10100101
```

```
         consensus w.r.t. variable 1: 01010000
         smoothing w.r.t. variable 1: 11110101

positive cofactor w.r.t. variable 2: 11011101
negative cofactor w.r.t. variable 2: 00010001
        derivative w.r.t. variable 2: 11001100
         consensus w.r.t. variable 2: 00010001
         smoothing w.r.t. variable 2: 11011101
```

The grading will be done on the *selsrv1* server, so please make sure your code compiles and runs well there. Please submit `cofactors.hpp` (without renaming it), compressed in a ZIP file, to Moodle. Note that other files will be replaced by the TA's version during grading.