1 Boolean Functions (45 points)

A Boolean function $f(x_1, \ldots, x_n)$ is said to be *positive unate* in input variable x_i if changing x_i from 0 to 1, while keeping the other inputs constant at any of their possible values, can never make f go from 1 to 0. In other words,

$$f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) \geq f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n)$$
$$\forall \; x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$$

Similarly, a Boolean function is *negative unate* in input variable x_i if changing x_i from 0 to 1 can never make f go from 0 to 1. A function that is neither positive unate or negative unate in an input variable is said to be *binate* in that variable.

1.1 Determining unateness (20 points)

For each of its input variables, determine whether the following function is positive unate, negative unate, or binate in it. Justify your answer.

$$f(v, w, x, y, z) = xy + wxz + vyz + wxyz + vwz + wy\bar{z} + w\bar{x}yz \tag{1}$$

1.2 Application of unateness (25 points)

A function that is unate in all of its input variables is simply called a *unate function*. Note that a unate function may be positive unate in some of its input variables and negative unate in the rest of its input variables. Unate functions are interesting because several operations on Boolean functions are simplified for the special case of unate functions. For this question, we will focus on *tautology checking*.

A Boolean function is a tautology if it evaluates to 1 for all possible inputs, *i.e.*, it is the constant 1 function. Checking whether a given function is a tautology is difficult in general. but it is much easier to do so for unate functions.

How can you check if a unate function given to you is a tautology? To receive credit, your answer must exploit unateness.

Write in Exam Book Only

2 Combinational Logic (55 points)

2.1 Two-level vs. multi-level implementations (20 points)

Give an example of a function of n inputs for which (i) the *minimum* sum-of-products implementation contains a number of product terms that is exponential in n, and (ii) a multi-level implementation exists that uses only 2-input AND gates, 2-input OR gates, and inverters, such that the total number of gates is polynomial in n. Justify the sizes of the two-level and multi-level implementations.

For this question, you can define a minimum sum-of-products implementation as one that has the lowest possible number of product terms among all possible sum-of-products implementations.

2.2 Multi-level minimization (10+15+10 = 35 points)

Factoring is an operation that can be used to convert two-level implementations into multi-level implementations. Given a function F, P is a factor of F if F can be re-written as $P.Q + R$, where Q is the quotient and R is the remainder. Ignore the trivial cases ($P = 0$, $P = 1$, $Q = 0$, and $Q = 1$).

For example, for $F = ab + bc + d$, $P = a + c$, $Q = b$, and $R = d$ is one possible way to factor F. Therefore, F can be re-written as $F = (a + c)b + d$. This expression for F cannot be factored any further.

Consider the following sum-of-products expression

$$Y = abg + acg + adf + aef + afg + bd + be + cd + ce \tag{2}$$

1. Create a new expression for Y by finding a sub-expression P that can be factored out of the sum-of-products expression. If there are multiple choices for P, pick any one.
2. Repeat the above process by recursively factoring the new expression for Y until no further factoring is possible (except for the trivial cases described above). Again, if there are multiple factors possible at any step, you may pick any one of them.
3. Based on the fully factored expression for Y, draw a multi-level circuit that uses only 2-input AND gates, 2-input OR gates, and inverters.

Write in Exam Book Only