

LAB 3

```
from itertools import combinations

def binary_str(n, length):
    return format(n, f'0{length}b')

def generate_perfect_terms(truth_table, n_vars, disjunctive=True):
    perfect_terms = []
    for i, val in enumerate(truth_table):
        if (disjunctive and val=='1') or (not disjunctive and val=='0'):
            perfect_terms.append(binary_str(i, n_vars))
    return perfect_terms

def reduce_terms(perfect_terms):
    groups = {}
    for term in perfect_terms:
        ones = term.count('1')
        groups.setdefault(ones, []).append(term)
    reduced = set()
    used = set()
    max_group = max(groups.keys()) if groups else 0
    for i in range(max_group+1):
        for t1 in groups.get(i, []):
            for t2 in groups.get(i+1, []):
                diff = [j for j,(a,b) in enumerate(zip(t1,t2)) if a!=b]
                if len(diff) == 1:
                    new_term = list(t1)
                    new_term[diff[0]] = '-'
                    reduced.add("".join(new_term))
                    used.add(t1)
                    used.add(t2)
    for term in perfect_terms:
        if term not in used:
            reduced.add(term)
    return sorted(reduced)

def build_coverage_table(reduced, perfect):
    table = {}
    for r in reduced:
        covers = []
        for p in perfect:
            match = True
            for rc, pc in zip(r, p):
                if rc != '-' and rc != pc:
                    match = False
                    break
            if match:
                covers.append(p)
        table[r] = covers
```

```

        match = False
        break
    if match:
        covers.append(p)
    table[r] = covers
return table

def find_essential_terms(table, perfect):
    essential = []
    covered = set()
    for p in perfect:
        count = sum(p in v for v in table.values())
        if count == 1:
            for r,v in table.items():
                if p in v:
                    if r not in essential:
                        essential.append(r)
                    covered.update(v)
    return essential, covered

def minimize_normal_form(truth_table, n_vars):
    print(f"Input truth table: {truth_table}\n")
    perfect_terms = generate_perfect_terms(truth_table, n_vars)
    print(f"Perfect terms (codified): {perfect_terms}\n")
    reduced_terms = reduce_terms(perfect_terms)
    print(f"Reduced terms (codified with '-'): {reduced_terms}\n")
    coverage_table = build_coverage_table(reduced_terms, perfect_terms)
    print("Coverage table:")
    for r,v in coverage_table.items():
        print(f"{r} -> {v}")
    essential, covered = find_essential_terms(coverage_table, perfect_terms)
    print(f"\nEssential reduced terms: {essential}\n")
    remaining = set(perfect_terms) - covered
    minimal_form = list(essential)
    for r in reduced_terms:
        if any(p in remaining for p in coverage_table[r]):
            minimal_form.append(r)
            remaining -= set(coverage_table[r])
    print(f"Minimal normal form (codified): {minimal_form}\n")

# Example usage:
truth_table_4var = "0001001101011110"
minimize_normal_form(truth_table_4var, 4)

```