Title:

Minimization of Boolean Functions using Quine-McCluskey Method

Submitted by:

NAME: NARAYAN PAUL

ROLL NO:23CS8040

SUBJECT: DIGITAL LOGIC DESIGN

SUBJECT CODE:CSC 302

Submitted to:

Dr. Bibhash Sen Sir

Department of Computer Science Engineering,

NIT Durgapur

Date:

03/09/2024

Acknowledgment

I would like to express my sincere gratitude to Dr. Bibhash Sen Sir for his invaluable guidance and support in the successful completion of this project. His insightful lectures on Digital Logic Design have greatly enhanced my understanding of the subject, and his encouragement throughout the project has been instrumental in its completion.

Table of Contents:

1. Introduction

The Quine-McCluskey method is a technique used for minimizing Boolean functions. It is particularly useful for simplifying functions in digital logic design and computer engineering. This report provides a detailed overview of the implementation of the Quine-McCluskey algorithm using Python.

2. Objective

The objective of this project is to implement the Quine-McCluskey algorithm in Python to minimize Boolean functions and generate the simplest possible expression for a given set of minterms and don't care terms.

3. Algorithm

Step 1: Input Minterms and Don't Cares

The user is prompted to input the minterms and don't care terms. These are converted into binary form based on the specified bit size.

Step 2: Grouping

The binary minterms are grouped based on the number of '1's in their binary representation.

Step 3: First Minimization

Adjacent groups are compared, and if only one bit differs, they are combined to form a new group with a dash ('_') representing the differing bit.

Step 4: Iterative Minimization

The process is repeated for subsequent groups until no further combinations are possible.

Step 5: Prime Implicant Chart

A Prime Implicant (PI) chart is generated, displaying the coverage of each minterm by the prime implicants.

Step 6: Essential Prime Implicants

Essential Prime Implicants (EPIs) are identified from the PI chart as those that cover a minterm uniquely.

Step 7: Minimization and Dominance

Row and column dominance are applied to further minimize the expression. If necessary, a semi-cyclic method is applied to select the lowest-cost prime implicant.

Step 8: Final Expression

The final minimized Boolean expression is generated and displayed.

4. Program Implementation

Input/Output

- **Input:**
  - Minterms: A list of minterms (in decimal).
  - Don't Cares: A list of don't care terms (in decimal).
  - Bit Size: The number of bits for the binary representation.
- **Output:**
  - The minimized Boolean expression.

1. Input Section

```
minterms = input("Write minterms: ").split()

dontcare = input("Write dontcares(if not there enter nothing): ").split()

print("==============Minterms================")

print(minterms)
```

```
print("==============Don't Cares================")

print("Don't Care: ",dontcare)

bitSize = int(input("Enter bitsize: "))
```

Input Handling:

- `minterms`: This takes the minterms from the user as a space-separated string and converts it into a list of strings.
- `dontcare`: Similarly, this takes the don't care conditions from the user and converts them into a list of strings. If the user does not provide any don't cares, it will be an empty list.
- **Print Statements**: These print the input values for minterms and don't cares to the console.
- `bitSize`: This stores the number of bits required for the binary representation of the minterms.

2. Initialization of Variables

```
unvisited = []

visited = []

list3 = []

dict1 = {}

dict2 = {}

max_space = 0

pi_chart = []

epi = []
```

```
epi_storage = []

column = [int(m) for m in minterms]
```

`unvisited` **and** **`visited`**: These lists keep track of minterms that have and haven't been combined during the minimization process.

`list3`, **`dict1`**, **`dict2`**: These are auxiliary variables used during the minimization process. `dict1` and `dict2` are used to store binary minterms grouped by the number of `1`s in their binary representation.

`max_space`: This variable keeps track of the maximum number of columns required in the PI chart, helping in formatting the output.

`pi_chart`: This list will store the Prime Implicant chart matrix.

`epi` **and** **`epi_storage`**: These lists store Essential Prime Implicants (EPIs) identified during the minimization process.

`column`: This is a list of minterms in decimal form, used as the column headers in the PI chart.

3. Converting Minterms to Binary

```
minterms_dontcare = []

minterms_dontcare = minterms + dontcare


def decimal_to_binary(minterm, bits=bitSize):

    # Convert the decimal number to binary and remove the '0b' prefix

    return format(int(minterm), f'0{bits}b')
```

```
binary_minterms = [decimal_to_binary(m, bitSize) for m in minterms_dontcare]
```

Combining Minterms and Don't Cares: `minterms_dontcare` is a list that combines both the minterms and don't cares for processing.

`decimal_to_binary` Function: Converts a decimal number to its binary representation with the specified bit size. This is used to convert each minterm and don't care condition into binary.

`binary_minterms`: This list stores the binary representation of all minterms and don't cares.

4. Grouping Minterms by Number of 1s

python

Copy code

```
dict1 = {(i,): [] for i in range(bitSize+1)}


def count1(s):

    s = str(s)

    return s.count('1')


# Populate the dictionary based on the count of '1's

for i in binary_minterms:

    key = (count1(i),)

    dict1[key].append(i)
```

- **`dict1` Initialization**: Creates a dictionary with keys representing the count of `1`s in the binary representation, ranging from `0` to `bitSize`.
- **`count1` Function**: This function counts the number of `1`s in a binary string.
- **Dictionary Population**: The binary minterms are stored in `dict1` under keys corresponding to their count of `1`s.

5. Printing the Grouped Minterms

```
print(f"   Group   ".ljust(40), end="")

print("     Binary numbers")

for key in sorted(dict1.keys()):

    a, = key

    print(f'    {str(a).ljust(40)}: ', end="")

    for j in dict1[key]:

        print(f'{j}, ', end="")

    print()  # New line after printing all minterms for a specific key
```

- **Print Group and Binary Numbers**: This section prints out the grouped binary minterms based on the number of `1`s in their binary representation.

6. Comparing Minterms for Minimization

```
def comparator(s1, s2):
```

```
    s1 = str(s1)

    s2 = str(s2)

    c = 0

    k = 0

    for i in range(len(s1)):

        if s1[i] != s2[i]:

            c = c + 1

            if c > 1:

                break

            if c == 1:

                k = i

    return c, k
```

- `comparator` **Function**: Compares two binary strings and returns:
  - `c`: The number of differing bits between the two strings.
  - `k`: The index of the differing bit, if there is only one difference.

7. Grouping Minterms After First Minimization Step

```
def binary_to_decimal(binary_str):

    return int(binary_str, 2)
```

```
keys_list = list(dict1.keys())

for i in range(len(keys_list) - 1):

    for m in dict1[keys_list[i]]:

        for n in dict1[keys_list[i + 1]]:

            a, b = comparator(m, n)

            if a == 1:

                list2 = list(m)

                list2[b] = '_'

                new_string = ''.join(list2)

                a = binary_to_decimal(m)

                b = binary_to_decimal(n)

                if a not in visited:

                    visited.append((a,))

                if b not in visited:

                    visited.append((b,))

                dict2[(a, b)] = []

                dict2[(a, b)].append(new_string)
```


- **Binary to Decimal Conversion**: Converts a binary string back to its decimal representation.
- **First Minimization Step**: This compares binary strings in adjacent groups (based on the number of 1s) to identify and

merge minterms that differ by only one bit. The resulting merged minterm (with a _ in place of the differing bit) is stored in `dict2`.

8. Printing the New Groups After Minimization

```
print("==========================================")

for i in dict2:

    print(f"{str(i).ljust(40)}", end="")

    print(dict2[i])
```

- **Print Merged Groups**: This prints out the newly formed groups after the first step of minimization.

9. Adding Unvisited Minterms

```
for i in minterms_dontcare:

    i = int(i)

    j = (i,)

    if j not in visited:

        k, = j

        binary_str = decimal_to_binary(int(k), bitSize)

        unvisited.append(((k,), binary_str))
```

- **Add Unvisited Minterms**: This section adds minterms that have not been merged during the first step of minimization to the `unvisited` list.

10. Recursive Minimization

```
def minimization():
    global max_space, unvisited
    dict1.clear()
    dict1.update(dict2)
    dict2.clear()

    keys_list = list(dict1.keys())
    for i in range(len(keys_list) - 1):
        for j in range(i + 1, len(keys_list)):
            for m in dict1[keys_list[i]]:
                for n in dict1[keys_list[j]]:
                    a, b = comparator(m, n)
                    if a == 1:
                        list2 = list(m)
                        list2[b] = '_'
                        new_string = ''.join(list2)

                        if keys_list[i] not in visited:
                            visited.append(keys_list[i])
                        if keys_list[j] not in visited:
                            visited.append(keys_list[j])
```

```
                        list3 = list(keys_list[i] + keys_list[j])
                        list3.sort()

                        tuple_key = tuple(list3)
                        if tuple_key not in dict2:
                            dict2[tuple_key] = []
                        if new_string not in dict2[tuple_key]:
                            dict2[tuple_key].append(new_string)
    for i in dict1:
        if i not in visited:
            for value in dict1[i]:
                unvisited.append((i, value))
                max_space = max(max_space, len(i))
```

- **Recursive Minimization**: This function performs additional minimization steps by comparing and merging binary strings across different groups, updating `dict1`, `dict2`, and `unvisited` as needed.

11. Printing Prime Implicants

```
for i in range(bitSize - 2):
    minimization()
print(f"Prime Implicants".ljust(40), end="")
print("Binary value")
```

```
for i, j in unvisited:
    print(f"   {str(i).ljust(40)}", end="")
    print(f"{j}")
```

- **Printing Prime Implicants**: After the minimization process, this section prints the prime implicants that were not combined further.

12. Generating the PI Chart Matrix

```
max_space = max_space + 1
pi_chart = [[' ' for j in range(len(column))] for i in range(len(unvisited))]


for i in range(len(unvisited)):
    for j in range(len(column)):
        if column[j] in unvisited[i][0]:
            pi_chart[i][j] = 'X'
```

- **PI Chart Matrix Generation**: This creates the Prime Implicant chart matrix, marking with an 'X' where a prime implicant covers a specific minterm.

13. Printing the PI Chart

```
print("======================================================")
print("Prime Implicants chart")
```

```
for i in range(len(pi_chart)):
    print(f"{str(unvisited[i][0]).ljust(40)}", end="")
    for j in range(len(pi_chart[i])):
        print(pi_chart[i][j], end="")
    print()
```

- **Print PI Chart**: This prints the Prime Implicant chart in a readable format.

14. Finding Essential Prime Implicants

```
for j in range(len(column)):
    c = 0
    pos = -1
    for i in range(len(pi_chart)):
        if pi_chart[i][j] == 'X':
            c += 1
            pos = i
    if c == 1:
        epi.append(unvisited[pos][0])
        epi_storage.append(unvisited[pos][1])


for i in range(len(pi_chart)):
    if unvisited[i][0] in epi:
        for j in range(len(column)):
```

```
            pi_chart[i][j] = ' '
```

- **Identify EPIs**: This section identifies the Essential Prime Implicants (EPIs) by checking for columns with only one 'X', indicating that a particular prime implicant is essential. EPIs are stored in `epi` and `epi_storage`.

15. Printing the Essential Prime Implicants

```
print("=====================================================")

print("EPI")

for i in range(len(epi)):

    print(f"   {str(epi[i]).ljust(40)}: {epi_storage[i]}")

print("=====================================================")
```

- **Print EPIs**: This prints the list of identified Essential Prime Implicants and their corresponding binary representations.

16. Dominance Check Function

```
def check_dominance():

    rows_to_remove = set()

    cols_to_remove = set()


    # Row Dominance

    for i in range(len(pi_chart)):

        for j in range(i + 1, len(pi_chart)):
```

```
            if all(pi_chart[i][k] == 'X' or pi_chart[i][k] == pi_chart[j][k] for k in range(len(column))):
                rows_to_remove.add(j)
            elif all(pi_chart[j][k] == 'X' or pi_chart[j][k] == pi_chart[i][k] for k in range(len(column))):
                rows_to_remove.add(i)


    # Column Dominance
    for i in range(len(column)):
        for j in range(i + 1, len(column)):
            if all(pi_chart[k][i] == 'X' or pi_chart[k][i] == pi_chart[k][j] for k in range(len(pi_chart))):
                cols_to_remove.add(j)
            elif all(pi_chart[k][j] == 'X' or pi_chart[k][j] == pi_chart[k][i] for k in range(len(pi_chart))):
                cols_to_remove.add(i)


    return list(rows_to_remove), list(cols_to_remove)
```

- **`check_dominance` Function**: This function checks for dominance in both rows and columns of the PI chart:
    - **Row Dominance**: If one row fully covers another row (meaning all 'X's in one row are also present in another), the dominated row is marked for removal.

- ○ **Column Dominance**: Similarly, if one column is fully covered by another, the dominated column is marked for removal.
- ○ **Return Values**: The function returns two lists, `rows_to_remove` and `cols_to_remove`, containing the indices of rows and columns that should be removed.

17. Removing Dominated Rows and Columns

```
def remove_dominated_rows_and_columns(rows_to_remove,
cols_to_remove):

    global pi_chart, column



    # Remove dominated rows

    pi_chart = [pi_chart[i] for i in range(len(pi_chart)) if i
not in rows_to_remove]



    # Remove dominated columns

    column = [column[i] for i in range(len(column)) if i not in
cols_to_remove]

    pi_chart = [[pi_chart[i][j] for j in range(len(pi_chart[i]))
if j not in cols_to_remove] for i in range(len(pi_chart))]
```

- **`remove_dominated_rows_and_columns` Function**:
  - ○ **Row Removal**: Removes rows from `pi_chart` that were marked for removal in `rows_to_remove`.

- **Column Removal**: Removes columns from `pi_chart` and `column` that were marked for removal in `cols_to_remove`.

18. Final Prime Implicant Selection

```
while True:
    rows_to_remove, cols_to_remove = check_dominance()
    if not rows_to_remove and not cols_to_remove:
        break
    remove_dominated_rows_and_columns(rows_to_remove, cols_to_remove)


print("=====================================")
print("Final PI chart after removing dominance")
for i in range(len(pi_chart)):
    print(f"{str(unvisited[i][0]).ljust(40)}", end="")
    for j in range(len(pi_chart[i])):
        print(pi_chart[i][j], end="")
    print()
```

- **Dominance Removal Loop**: This loop continues checking for and removing dominated rows and columns until no more dominance is detected.

- **Print Final PI Chart**: After all dominated rows and columns are removed, the final PI chart is printed, showing the remaining prime implicants and their coverage.

19. Selecting Final Essential Prime Implicants

```
if len(column) == 0:
    print("All minterms are covered by Essential Prime Implicants.")
else:
    for i in range(len(pi_chart)):
        if 'X' in pi_chart[i]:
            epi.append(unvisited[i][0])
            epi_storage.append(unvisited[i][1])


print("====================================================")
print("Final Essential Prime Implicants (after dominance)")
for i in range(len(epi)):
    print(f"   {str(epi[i]).ljust(40)}: {epi_storage[i]}")
print("====================================================")
```

- **Check for Full Coverage**: If all columns are removed, it means all minterms are covered by the Essential Prime Implicants (EPIs).
- **Select Remaining EPIs**: If there are still uncovered minterms, the remaining rows in the PI chart are considered EPIs.

- **Print Final EPIs**: The final list of EPIs is printed after the dominance check, providing the minimum set of prime implicants required to cover all minterms.

Full Code:

```
# Input minterms and split them into a list

minterms = input("Write minterms: ").split()

dontcare = input("Write dontcares(if not there enter nothing):
").split()

print("===============Minterms=================")

print(minterms)

print("===============Don't Cares=================")

print("Don't Care: ",dontcare)

bitSize = int(input("Enter bitsize: "))

unvisited=[]

visited=[]

unvisited=[]

list3=[]

dict1={}

dict2={}

max_space=0

pi_chart=[]

epi=[]
```

```
epi_storage=[]

column=[int(m) for m in minterms]

# Convert minterms to binary

minterms_dontcare=[]

minterms_dontcare=minterms+dontcare

def decimal_to_binary(minterm, bits=bitSize):

    # Convert the decimal number to binary and remove the '0b'
prefix

    return format(int(minterm), f'0{bits}b')



binary_minterms = [decimal_to_binary(m,bitSize) for m in
minterms_dontcare]



# Initialize dictionary to store minterms based on the number of
'1's in their binary representation



dict1 = {(i,): [] for i in range(bitSize+1)}



def count1(s):

    s=str(s)

    return s.count('1')
```

```
# Populate the dictionary based on the count of '1's

for i in binary_minterms:

    key = (count1(i),)

    dict1[key].append(i)



# Print the contents of the dictionary

print(f"   Group   ".ljust(40),end="")

print("     Binary numbers")

for key in sorted(dict1.keys()):

    a,=key

    print(f'    {str(a).ljust(40)}: ', end="")

    for j in dict1[key]:

        print(f'{j}, ', end="")

    print()  # New line after printing all minterms for a
specific key



def comparator(s1,s2):

    s1=str(s1)

    s2=str(s2)

    c=0

    k=0

    for i in range(len(s1)):
```

```
        if(s1[i]!=s2[i]):

            c=c+1

            if(c>1):

                break

            if(c==1):

                k=i

    return c,k


def binary_to_decimal(binary_str):

    return int(binary_str, 2)


keys_list=list(dict1.keys())

for i in range(len(keys_list)-1):

    for m in dict1[keys_list[i]]:

        for n in dict1[keys_list[i+1]]:

            a,b=comparator(m,n);

            if(a==1):

                list2=list(m)

                list2[b]='_'

                new_string=''.join(list2)

                a=binary_to_decimal(m)
```

```
                b=binary_to_decimal(n)
                if a not in visited:
                    visited.append((a,))
                if b not in visited:
                    visited.append((b,))
                dict2[(a,b)]=[]
                dict2[(a,b)].append(new_string)
print("==========================================")
for i in dict2:
    print(f"{str(i).ljust(40)}",end="")
    print(dict2[i])
for i in minterms_dontcare:
    i=int(i)
    j=(i,)
    if j not in visited:
        k,=j
        binary_str = decimal_to_binary(int(k), bitSize)
        unvisited.append(((k,), binary_str))
def minimization():
    global max_space,unvisited
    # Ensure dict1 is updated from the previous phase
```

```
    dict1.clear()

    dict1.update(dict2)

    dict2.clear()


    keys_list = list(dict1.keys())


    # Third level minimization

    for i in range(len(keys_list) - 1):

        for j in range(i + 1, len(keys_list)):

            for m in dict1[keys_list[i]]:

                for n in dict1[keys_list[j]]:

                    a, b = comparator(m, n)

                    if a == 1:

                        list2 = list(m)

                        list2[b] = '_'

                        new_string = ''.join(list2)


                        if keys_list[i] not in visited:

                            visited.append(keys_list[i])

                        if keys_list[j] not in visited:

                            visited.append(keys_list[j])
```

```
                        # Ensure list3 is initialized
                        list3 = list(keys_list[i] +
keys_list[j])
                        list3.sort()

                        # Use tuple of sorted keys as dict key
                        tuple_key = tuple(list3)
                        if tuple_key not in dict2:
                            dict2[tuple_key] = []
                        if new_string not in dict2[tuple_key]:
                            dict2[tuple_key].append(new_string)
    for i in dict1:
        if i not in visited:
            for value in dict1[i]:
                unvisited.append((i, value))
                max_space=max(max_space,len(i))
    print("==========================================")
    for i in dict2:
        print(f"{str(i).ljust(40)}",end="")
        print(dict2[i])
```

```
for i in range(bitSize-2):

    minimization()

print(f"Prime Implicants".ljust(40),end="")

print("Binary value")

for i,j in unvisited:

    print(f"   {str(i).ljust(40)}",end="")

    print(f"{j}")

# print(max_space)

# Populate the PI chart matrix

def PIchart(column,unvisited):

    global pi_chart

    pi_chart=[]

    for row in unvisited:

        pi_row = []

        minterm_tuple, binary_string = row

        for minterm in column:

            if minterm in minterm_tuple:

                pi_row.append("X")

            else:

                pi_row.append(" ")

        pi_chart.append(pi_row)
```

```
    # Print the PI chart matrix
def print_PI_chart(pi_chart,unvisited,column):
    width = 4 * max_space

print("=======================================================")
    print("PI Chart Matrix:")

print("=======================================================")
    headers = []
    a=False
    for i in column:
        if(a==False):
            headers.append(str(i))  # Remove the extra quotes
and space
            a=True
        elif i < 10:
            headers.append("   " + str(i))
        else:
            headers.append("  " + str(i))
    header = "".join(headers)
```

```
    print(f"{' ' * width}:{header}")

    for idx, row in enumerate(pi_chart):

        minterm_tuple, binary_string = unvisited[idx]

        print(f"{str(minterm_tuple).ljust(width)}:{'
'.join(row)}")


def find_epi(pi_chart, column):

    epi_indices = []

    epi_minterms = []


    for col_idx in range(len(column)):

        x_count = 0

        last_row_with_x = -1

        for row_idx in range(len(pi_chart)):

            if pi_chart[row_idx][col_idx] == 'X':

                x_count += 1

                last_row_with_x = row_idx


        # If only one 'X' is found in the column, it's an EPI

        if x_count == 1:

            epi_indices.append(last_row_with_x)

            epi_minterms.append(column[col_idx])
```

```
    return epi_indices, epi_minterms


def is_set_empty(my_set):
    return not bool(my_set)


def apply_row_dominance(pi_chart, unvisited):
    rows_to_remove = []

    # Compare each pair of rows to find dominated rows
    for i in range(len(pi_chart)):
        for j in range(i + 1, len(pi_chart)):
            # Get 'X' positions in row i
            x_positions_i = []
            for k in range(len(pi_chart[i])):
                if pi_chart[i][k] == 'X':
                    x_positions_i.append(k)

            # Get 'X' positions in row j
            x_positions_j = []
            for k in range(len(pi_chart[j])):
```

```
                if pi_chart[j][k] == 'X':
                    x_positions_j.append(k)

            # Check if row i is dominated by row j
            is_i_subset_j = True
            for pos in x_positions_i:
                if pos not in x_positions_j:
                    is_i_subset_j = False
                    break
            if is_i_subset_j:
                rows_to_remove.append(i)
                continue  # Move to the next pair after marking
i for removal

            # Check if row j is dominated by row i
            is_j_subset_i = True
            for pos in x_positions_j:
                if pos not in x_positions_i:
                    is_j_subset_i = False
                    break
            if is_j_subset_i:
                rows_to_remove.append(j)
```

```
    # Remove duplicates by converting to a set
    unique_rows_to_remove = set(rows_to_remove)
    if(is_set_empty(unique_rows_to_remove)):
        return -1,-1
    print("==================Rows removed:=================")
    for i in unique_rows_to_remove:
        print(unvisited[i])

    # Create new lists excluding the rows to remove
    new_pi_chart = []
    new_unvisited = []
    for index in range(len(pi_chart)):
        if index not in unique_rows_to_remove:
            new_pi_chart.append(pi_chart[index])
            new_unvisited.append(unvisited[index])

    return new_pi_chart, new_unvisited

def column_dominance(pi_chart):
    global column, unvisited
```

```
    columns_to_remove = set()


    num_rows = len(pi_chart)

    num_cols = len(pi_chart[0]) if num_rows > 0 else 0


    # Iterate over each pair of columns

    for i in range(num_cols-1):

        # for j in range(i + 1, num_cols):  # Start j from i+1
to avoid comparing the same pair twice

            # Check if column i is dominated by column j

            dominated = True

            for row in range(num_rows):

                if pi_chart[row][i] == 'X' and
pi_chart[row][i+1] != 'X':

                    dominated = False

                    break

            if dominated:

                columns_to_remove.add(i)

                continue  # Move to the next pair after marking
i for removal


            # Check if column j is dominated by column i
```

```
            dominated = True

            for row in range(num_rows):

                if pi_chart[row][i+1] == 'X' and pi_chart[row][i] != 'X':

                    dominated = False

                    break

            if dominated:

                columns_to_remove.add(i)

    if not columns_to_remove:

        return -1  # No columns were removed

    print("=================Columns deleted:=================")

    for i in sorted((columns_to_remove)):

        print(column[i])

    # Create new column and pi_chart lists excluding the columns to remove

    new_column = [column[i] for i in range(num_cols) if i not in columns_to_remove]

    print(new_column)

    new_pi_chart = []

    # Update the global variables

    column.clear()

    column = new_column.copy()
```

```
    PIchart(column, unvisited)  # Recreate the PI chart with updated columns

    new_pi_chart=pi_chart[:]

    return new_pi_chart




PIchart(column,unvisited)

print_PI_chart(pi_chart,unvisited,column)

print("============================================")


def stringtoexpression(s):

    s=str(s)

    list1=list(s)

    list2=[]

    for i in range(len(list1)):

        if list1[i]=='1':

            list2.append(chr(i+65))

        elif(list1[i]=='0'):

            list2.append(chr(i+65)+"'")

    new_string=''.join(list2)

    return new_string
```

```
def calculate_pi_cost(pi, minterm_coverage):
    num_literals = pi.count('1') + pi.count('0')  # Count of literals ('0' or '1')
    return num_literals / len(minterm_coverage)


def select_lowest_cost_pi(pi_chart, unvisited, column):
    min_cost = float('inf')
    selected_pi_index = -1


    for i, (minterm_tuple, binary_string) in enumerate(unvisited):
        covered_minterms = [column[j] for j in range(len(column)) if pi_chart[i][j] == 'X']
        cost = calculate_pi_cost(binary_string, covered_minterms)


        if cost < min_cost:
            min_cost = cost
            selected_pi_index = i


    return selected_pi_index


def minimization2():
```

```
    global pi_chart,column,unvisited

    list5=[]

    a1,b1=find_epi(pi_chart,column)

    if  len(a1)==0:

        print("The EPI is empty")

new_pi_chart,new_unvisited=apply_row_dominance(pi_chart,unvisite
d)

        if(new_pi_chart==-1):

            print("Row dominance not applied")

            # return -1

            new_pi_chart=[]

            new_pi_chart=column_dominance(pi_chart)

            if(new_pi_chart==-1):

                print("Column dominance not applied")

                new_column=[]

                new_unvisited=[]

a=select_lowest_cost_pi(pi_chart,unvisited,column)

                print("Semicyclic method applied")

                print("Selected PI: ",end="")

                print(unvisited[a])
```

```
            for i in range(len(unvisited)):
                if i!=a:
                    new_unvisited.append(unvisited[i])
            for i in column:
                if i!=unvisited[a]:
                    new_column.append(i)
            if len(new_column) == 0:
                print("All minterms are covered. Exiting.")
                return 0
            column.clear()
            unvisited.clear()
            column=new_column.copy()
            unvisited=new_unvisited.copy()
            PIchart(column,unvisited)
            print_PI_chart(pi_chart,unvisited,column)
            return 1
        else:
            pi_chart.clear()
            pi_chart=new_pi_chart[:]
            print_PI_chart(pi_chart,unvisited,column)
            return 1
```

```
        pi_chart.clear()

        unvisited.clear()

        pi_chart=new_pi_chart[:]

        unvisited=new_unvisited.copy()

        print_PI_chart(pi_chart,unvisited,column)

        return 1

    epi_string,epi_minterms=find_epi(pi_chart,column)

    for i in epi_string:

        a,b=unvisited[i]

        epi_storage.append(unvisited[i])

        list5.extend(a)


    new_column=[]

    new_unvisited=[]

    for i in range(len(unvisited)):

        if i not in epi_string:

            new_unvisited.append(unvisited[i])

    for i in column:

        if i not in list5:

            new_column.append(i)

    if len(new_column) == 0:
```

```
        print("All minterms are covered. Exiting.")
        return 0
    column.clear()
    unvisited.clear()
    column=new_column.copy()
    unvisited=new_unvisited.copy()
    PIchart(column,unvisited)
    print_PI_chart(pi_chart,unvisited,column)
    return 1
abc=1
while(abc!=0):
    abc=minimization2()
    print("==================================================")
if len(epi_storage)==0:
    epi_storage=unvisited.copy()
result=[]
b=epi_storage
for i in b:
    r,s=i
    result.append(stringtoexpression(s))
set1={}
```

```
set1=set(result)

list3=list(set1)

print("=============================================")

print("Final Expression: ")

i=0

for i in range(len(list3)-1):

    print(str(list3[i]),'+ ',end="")

print(str(list3[i+1]))



# Input:

# 0 1 2 8 9 15 17 21 24 25 27 31

# 0 2 4 5 8 10 12 13 18 21 22 23 25 26 27 29

# 1 3 5 7  9 11 13 15 20 21 22 23 28 29 30 31 36 37 38 39 44 45
46 47 49 51 53 55 57 59 61 63
```

5. Test Cases and Outputs

Test Case 1:

- **Input:**
    - Minterms: 0 1 2 8 9 15 17 21 24 25 27 31
    - Don't Cares: None
    - Bit Size: 5
- **Output:**

```
Write minterms: 0 1 2 8 9 15 17 21 24 25 27 31
Write dontcares(if not there enter nothing):
===============Minterms=================
['0', '1', '2', '8', '9', '15', '17', '21', '24', '25', '27', '31']
===============Don't Cares=================
Don't Care:  []
Enter bitsize: 5
   Group                                    Binary numbers
    0                                      : 00000,
    1                                      : 00001, 00010, 01000,
    2                                      : 01001, 10001, 11000,
    3                                      : 10101, 11001,
    4                                      : 01111, 11011,
    5                                      : 11111,
==========================================
(0, 1)                                   ['0000_']
(0, 2)                                   ['000_0']
(0, 8)                                   ['0_000']
(1, 9)                                   ['0_001']
(1, 17)                                  ['_0001']
(8, 9)                                   ['0100_']
(8, 24)                                  ['_1000']
(9, 25)                                  ['_1001']
(17, 21)                                 ['10_01']
(17, 25)                                 ['1_001']
(24, 25)                                 ['1100_']
(25, 27)                                 ['110_1']
(15, 31)                                 ['_1111']
(27, 31)                                 ['11_11']
==========================================
(0, 1, 8, 9)                             ['0_00_']
(1, 9, 17, 25)                           ['__001']
(8, 9, 24, 25)                           ['_100_']
==========================================
==========================================
Prime Implicants                         Binary value
   (0, 2)                                   000_0
   (17, 21)                                 10_01
```

```
(25, 27)        :                                    X   X
(15, 31)        :                    X                       X
(27, 31)        :                                        X   X
(0, 1, 8, 9)    :X   X       X   X
(1, 9, 17, 25)  :    X           X       X           X
(8, 9, 24, 25)  :            X   X               X   X
============================================
======================================================
PI Chart Matrix:
======================================================
                :1  27
(25, 27)        :    X
(27, 31)        :    X
(0, 1, 8, 9)    :X
(1, 9, 17, 25)  :X
==================================================
The EPI is empty
==================Rows removed:=================
((25, 27), '110_1')
((0, 1, 8, 9), '0_00_')
======================================================
PI Chart Matrix:
======================================================
                :1  27
(27, 31)        :    X
(1, 9, 17, 25)  :X
==================================================
All minterms are covered. Exiting.
==================================================
=============================================
Final Expression:
ABDE + BCDE + A'B'C'E' + AB'D'E + BC'D' + C'D'E
```

Test Case 2:

- **Input:**
  - Minterms: 0 1 2 5 6 7 8 9 10 14
  - Don't Cares: None
  - Bit Size: 4
- **Output:**

```
Write minterms:  0 1 2 5 6 7 8 9 10 14
Write dontcares(if not there enter nothing):
===============Minterms================
['0', '1', '2', '5', '6', '7', '8', '9', '10', '14']
===============Don't Cares=================
Don't Care:  []
Enter bitsize: 4
   Group                                   Binary numbers
    0                                     : 0000,
    1                                     : 0001, 0010, 1000,
    2                                     : 0101, 0110, 1001, 1010,
    3                                     : 0111, 1110,
    4                                     :
==========================================
(0, 1)                                 ['000_']
(0, 2)                                 ['00_0']
(0, 8)                                 ['_000']
(1, 5)                                 ['0_01']
(1, 9)                                 ['_001']
(2, 6)                                 ['0_10']
(2, 10)                                ['_010']
(8, 9)                                 ['100_']
(8, 10)                                ['10_0']
(5, 7)                                 ['01_1']
(6, 7)                                 ['011_']
(6, 14)                                ['_110']
(10, 14)                               ['1_10']
==========================================
(0, 1, 8, 9)                           ['_00_']
(0, 2, 8, 10)                          ['_0_0']
(2, 6, 10, 14)                         ['__10']
==========================================
Prime Implicants                       Binary value
   (1, 5)                                 0_01
   (5, 7)                                 01_1
   (6, 7)                                 011_
   (0, 1, 8, 9)                           _00_
```

```
   (0, 2, 8, 10)                            _0_0
   (2, 6, 10, 14)                           __10
======================================================
PI Chart Matrix:
======================================================
                :0   1   2   5   6   7   8   9   10  14
(1, 5)          :    X       X
(5, 7)          :            X       X
(6, 7)          :                X   X
(0, 1, 8, 9)    :X   X                   X   X
(0, 2, 8, 10)   :X       X               X       X
(2, 6, 10, 14)  :        X       X               X   X
============================================
======================================================
PI Chart Matrix:
======================================================
                :5   7
(1, 5)          :X
(5, 7)          :X   X
(6, 7)          :    X
(0, 2, 8, 10)   :
==================================================
The EPI is empty
==================Rows removed:=================
((1, 5), '0_01')
((6, 7), '011_')
((0, 2, 8, 10), '_0_0')
======================================================
PI Chart Matrix:
======================================================
                :5   7
(5, 7)          :X   X
==================================================
All minterms are covered. Exiting.
==================================================
=============================================
Final Expression:
A'BD + CD' + B'C'
```

```
Write minterms: 1 3 5 7  9 11 13 15 20 21 22 23 28 29 30 31 36 37 38 39 44 45 46 47 49 51 53 55 57 59 61 63
Write dontcares(if not there enter nothing):
===============Minterms=================
['1', '3', '5', '7', '9', '11', '13', '15', '20', '21', '22', '23', '28', '29', '30', '31', '36', '37', '38', '39', '44', '45', '46', '47', '49', '51',
53', '55', '57', '59', '61', '63']
===============Don't Cares=================
Don't Care:  []
Enter bitsize: 6
   Group                                        Binary numbers
    0                                          :
    1                                          : 000001,
    2                                          : 000011, 000101, 001001, 010100, 100100,
    3                                          : 000111, 001011, 001101, 010101, 010110, 011100, 100101, 100110, 101100, 110001,
    4                                          : 001111, 010111, 011101, 011110, 100111, 101101, 101110, 110011, 110101, 111001,
    5                                          : 011111, 101111, 110111, 111011, 111101,
    6                                          : 111111,
==========================================
(1, 3)                                      ['0000_1']
(1, 5)                                      ['000_01']
(1, 9)                                      ['00_001']
(3, 7)                                      ['000_11']
(3, 11)                                     ['00_011']
(5, 7)                                      ['0001_1']
(5, 13)                                     ['00_101']
(5, 21)                                     ['0_0101']
(5, 37)                                     ['_00101']
(9, 11)                                     ['0010_1']
(9, 13)                                     ['001_01']
(20, 21)                                    ['01010_']
(20, 22)                                    ['0101_0']
(20, 28)                                    ['01_100']
(36, 37)                                    ['10010_']
(36, 38)                                    ['1001_0']
(36, 44)                                    ['10_100']
(7, 15)                                     ['00_111']
(7, 23)                                     ['0_0111']
(7, 39)                                     [' 00111']
```

```
(11, 15)                                                    ['001_11']
(13, 15)                                                    ['0011_1']
(13, 29)                                                    ['0_1101']
(13, 45)                                                    ['_01101']
(21, 23)                                                    ['0101_1']
(21, 29)                                                    ['01_101']
(21, 53)                                                    ['_10101']
(22, 23)                                                    ['01011_']
(22, 30)                                                    ['01_110']
(28, 29)                                                    ['01110_']
(28, 30)                                                    ['0111_0']
(37, 39)                                                    ['1001_1']
(37, 45)                                                    ['10_101']
(37, 53)                                                    ['1_0101']
(38, 39)                                                    ['10011_']
(38, 46)                                                    ['10_110']
(44, 45)                                                    ['10110_']
(44, 46)                                                    ['1011_0']
(49, 51)                                                    ['1100_1']
(49, 53)                                                    ['110_01']
(49, 57)                                                    ['11_001']
(15, 31)                                                    ['0_1111']
(15, 47)                                                    ['_01111']
(23, 31)                                                    ['01_111']
(23, 55)                                                    ['_10111']
(29, 31)                                                    ['0111_1']
(29, 61)                                                    ['_11101']
(30, 31)                                                    ['01111_']
(39, 47)                                                    ['10_111']
(39, 55)                                                    ['1_0111']
(45, 47)                                                    ['1011_1']
(45, 61)                                                    ['1_1101']
(46, 47)                                                    ['10111_']
(51, 55)                                                    ['110_11']
(51, 59)                                                    ['11_011']
(53, 55)                                                    ['1101_1']
(53, 61)                                                    ['11_101']
(57, 59)                                                    ['1110_1']
(57, 61)                                                    ['111_01']
(31, 63)                                                    ['_11111']
```

```
(47, 63)                                        ['1_1111']
(55, 63)                                        ['11_111']
(59, 63)                                        ['111_11']
(61, 63)                                        ['1111_1']
==========================================
(1, 3, 5, 7)                                    ['000__1']
(1, 3, 9, 11)                                   ['00_0_1']
(1, 5, 9, 13)                                   ['00__01']
(3, 7, 11, 15)                                  ['00__11']
(5, 7, 13, 15)                                  ['00_1_1']
(5, 7, 21, 23)                                  ['0_01_1']
(5, 7, 37, 39)                                  ['_001_1']
(5, 13, 21, 29)                                 ['0__101']
(5, 13, 37, 45)                                 ['_0_101']
(5, 21, 37, 53)                                 ['__0101']
(9, 11, 13, 15)                                 ['001__1']
(20, 21, 22, 23)                                ['0101__']
(20, 21, 28, 29)                                ['01_10_']
(20, 22, 28, 30)                                ['01_1_0']
(36, 37, 38, 39)                                ['1001__']
(36, 37, 44, 45)                                ['10_10_']
(36, 38, 44, 46)                                ['10_1_0']
(7, 15, 23, 31)                                 ['0__111']
(7, 15, 39, 47)                                 ['_0_111']
(7, 23, 39, 55)                                 ['__0111']
(13, 15, 29, 31)                                ['0_11_1']
(13, 15, 45, 47)                                ['_011_1']
(13, 29, 45, 61)                                ['__1101']
(21, 23, 29, 31)                                ['01_1_1']
(21, 23, 53, 55)                                ['_101_1']
(21, 29, 53, 61)                                ['_1_101']
(22, 23, 30, 31)                                ['01_11_']
(28, 29, 30, 31)                                ['0111__']
(37, 39, 45, 47)                                ['10_1_1']
(37, 39, 53, 55)                                ['1_01_1']
(37, 45, 53, 61)                                ['1__101']
(38, 39, 46, 47)                                ['10_11_']
(44, 45, 46, 47)                                ['1011__']
(49, 51, 53, 55)                                ['110__1']
(49, 51, 57, 59)                                ['11_0_1']
```

```
(49, 53, 57, 61)                         ['11__01']
(15, 31, 47, 63)                         ['__1111']
(23, 31, 55, 63)                         ['_1_111']
(29, 31, 61, 63)                         ['_111_1']
(39, 47, 55, 63)                         ['1__111']
(45, 47, 61, 63)                         ['1_11_1']
(51, 55, 59, 63)                         ['11__11']
(53, 55, 61, 63)                         ['11_1_1']
(57, 59, 61, 63)                         ['111__1']
==========================================
(1, 3, 5, 7, 9, 11, 13, 15)              ['00___1']
(5, 7, 13, 15, 21, 23, 29, 31)           ['0__1_1']
(5, 7, 13, 15, 37, 39, 45, 47)           ['_0_1_1']
(5, 7, 21, 23, 37, 39, 53, 55)           ['__01_1']
(5, 13, 21, 29, 37, 45, 53, 61)          ['___101']
(20, 21, 22, 23, 28, 29, 30, 31)         ['01_1__']
(36, 37, 38, 39, 44, 45, 46, 47)         ['10_1__']
(7, 15, 23, 31, 39, 47, 55, 63)          ['___111']
(13, 15, 29, 31, 45, 47, 61, 63)         ['__11_1']
(21, 23, 29, 31, 53, 55, 61, 63)         ['_1_1_1']
(37, 39, 45, 47, 53, 55, 61, 63)         ['1__1_1']
(49, 51, 53, 55, 57, 59, 61, 63)         ['11___1']
==========================================
(5, 7, 13, 15, 21, 23, 29, 31, 37, 39, 45, 47, 53, 55, 61, 63)['___1_1']
==========================================
Prime Implicants                         Binary value
   (1, 3, 5, 7, 9, 11, 13, 15)              00___1
   (20, 21, 22, 23, 28, 29, 30, 31)         01_1__
   (36, 37, 38, 39, 44, 45, 46, 47)         10_1__
   (49, 51, 53, 55, 57, 59, 61, 63)         11___1
   (5, 7, 13, 15, 21, 23, 29, 31, 37, 39, 45, 47, 53, 55, 61, 63)___1_1
======================================================
PI Chart Matrix:
======================================================
                                                        :1   3   5   7   9   11  13  15  20  21  22  23  28  29  30  31  36  37  38  39  44  45  4
6  47  49  51  53  55  57  59  61  63
(1, 3, 5, 7, 9, 11, 13, 15)                             :X   X   X   X   X   X   X   X

(20, 21, 22, 23, 28, 29, 30, 31)                        :                                X   X   X   X   X   X   X   X



(36, 37, 38, 39, 44, 45, 46, 47)                         :                                                                X   X   X   X   X   X
X   X
(49, 51, 53, 55, 57, 59, 61, 63)                         :
       X   X   X   X   X   X   X   X
(5, 7, 13, 15, 21, 23, 29, 31, 37, 39, 45, 47, 53, 55, 61, 63)  :     X   X       X   X     X     X     X     X     X     X     X
   X           X   X           X   X
============================================
All minterms are covered. Exiting.
==================================================
============================================
Final Expression:
ABF + AB'D + A'B'F + A'BD
```

6. Conclusion

The implementation of the Quine-McCluskey algorithm in Python effectively minimizes Boolean functions. The detailed step-by-step minimization process, along with the application of dominance and cost-based selection, ensures that the simplest possible expression is derived.

7. References

1. Class notes by Dr Bibhash Sen Sir
2. Digital Logic Design, M. Mano, M. D. Ciletti
3. Samuel C. Lee

Thank You