Digital System Design Project 2 – Boolean Function Minimization

B11330031 林羿里

1. Source Code Architecture

I. Overall Control (main.cpp)

int main(int argc, char* argv[])

- **功能**：程式主要入口點，負責整體流程控制。
- **流程**：
  1. 檢查並解析命令列參數（輸入 PLA 檔案路徑、輸出 PLA 檔案路徑）。
  2. 建立 PlaParser 物件，呼叫其 parse 方法讀取並解析輸入檔案。
  3. 建立 QuineMcCluskey 物件，執行演算法以生成所有的質蘊涵項 (Prime Implicants)。
  4. 建立 Petrick 物件，執行 Petrick's method 來找出成本最低的質蘊涵項覆蓋組合。
  5. 建立 PlaWriter 物件，將最小化後的結果寫入指定的輸出檔案，並在終端機顯示統計資訊。

II. PLA File Parser (PlaParser.h, PlaParser.cpp)

bool parse(const string& filename)

- **功能**：讀取並解析標準 PLA 格式檔案。
- **處理內容**：
  - .i (輸入變數數量)
  - .o (輸出變數數量，本專案限定為 1)

- .ilb (輸入變數名稱)
- .ob (輸出變數名稱)
- .p (積項數量)
- 積項 (Product terms) 及其對應的輸出 (1 代表 on-set, - 代表 don't care)。

const vector<int>& getMinterms() const

- **功能**：回傳從 PLA 檔案中解析出來的 on-set 最小項 (minterms) 列表。

const vector<int>& getDontCares() const

- **功能**：回傳 don't-care 最小項列表。

III. Quine-McCluskey Algorithm (QuineMcCluskey.h, QuineMcCluskey.cpp)

void generatePrimeImplicants(...)

- **功能**：從 on-set 與 don't-care 最小項中，生成所有的質蘊涵項。
- **演算法步驟**：
  1. **分組 (Grouping)**：根據二進位表示中 1 的數量，將所有最小項 (包含 on-set 與 don't care) 進行分組。
  2. **合併 (Combining)**：反覆比較相鄰組別的項，若僅有一個位元不同，則將其合併成一個更大的項，並在不同的位元標記為 don't care (-)。
  3. **標記 (Marking)**：在合併過程中，所有被成功合併的項都會被標記。
  4. **找出質蘊涵項**：迭代直到沒有任何項可以再合併為止。所有未被標記的項即為質蘊涵項。

IV. Petrick's Algorithm (Petrick.h, Petrick.cpp)

vector<Implicant> findMinimalCover(...)

- **功能**：應用 Petrick's method，從質蘊涵項中找出一個或多個成本最低的覆蓋解。
- **演算法步驟**：

1. **建立覆蓋表 (Coverage Chart)**：建立一個表格，列出所有質蘊涵項以及它們能覆蓋的 on-set 最小項。
2. **找出必要質蘊涵項 (Essential PIs)**：識別那些唯一覆蓋了某個最小項的質蘊涵項，這些是最終解的必要部分。
3. **建立布林表示式 (P-function)**：對於每個尚未被覆蓋的最小項，將所有能覆蓋它的質蘊涵項以 OR (+) 形式寫出。再將所有最小項的表示式以 AND (*) 形式連接起來。
4. **展開與化簡**：使用分配律 (A+B)(C+D) = AC+AD+BC+BD 將布林表示式展開成 SOP (Sum of Products) 形式。
5. **選擇最佳解**：每一個積項都代表一個可行的覆蓋解。計算每個解的成本（積項數量與文字數量），並選擇成本最低的解。

2. Test Cases

Test Case 1: 4-Variable Boolean Function

Input PLA File (pla_files/test1.pla)

```
.i 4
.o 1
.ilb A B C D
.ob F
.p 10
0000 1
0001 1
0010 1
0011 1
0100 -
0101 1
```

1000 1

1001 -

1100 1

1101 1

.e

Output PLA File (output_pla/test1_output.pla)

Minimized Boolean Function

Generated by Quine-McCluskey + Petrick's Algorithm

Statistics:

Product terms: 2

Total literals: 3

.i 4

.o 1

.ilb A B C D

.ob F

.p 2

00-- 1

--0- 1

.e

Test Case 2: 5-Variable Boolean Function

Input PLA File (pla_files/test2.pla)

```
.i 5
.o 1
.ilb A B C D E
.ob F
.p 15
00000 1
00001 1
00010 -
00100 1
00101 1
01000 1
01001 -
01100 -
10000 1
10001 1
10010 1
11000 1
11001 1
11100 -
11110 1
.e
```

Output PLA File (output_pla/test2_output.pla)

```
# Minimized Boolean Function
# Generated by Quine-McCluskey + Petrick's Algorithm
```

```
# Statistics:
#   Product terms: 4
#   Total literals: 12
.i 5
.o 1
.ilb A B C D E
.ob F
.p 4
111-0 1
00-0- 1
-00-0 1
--00- 1
.e
```

Test Case 3: 6-Variable Boolean Function

Input PLA File (pla_files/test3.pla)

```
.i 6
.o 1
.ilb A B C D E F
.ob F
.p 20
000000 1
000001 1
000010 1
000100 -
```

001000 1

001001 1

001100 -

010000 1

010001 1

010010 -

010100 1

100000 1

100001 1

100010 1

100100 -

101000 1

110000 1

110001 -

111000 1

111100 -

.e

Output PLA File (output_pla/test3_output.pla)

Minimized Boolean Function

Generated by Quine-McCluskey + Petrick's Algorithm

Statistics:

Product terms: 5

Total literals: 19

.i 6

```
.o 1
.ilb A B C D E F
.ob F
.p 5
00-00- 1
-000-0 1
0-0-00 1
1--000 1
--000- 1
.e
```

3. Compilation and Execution

Build Instructions (Linux/WSL)

```
# 建議先清除舊的編譯檔案
make clean

# 編譯程式碼以生成執行檔 'minimize'
make
```

Execution Example

可使用 make run 搭配目標 .pla 檔案名稱來執行，例如：

```
make run test1.pla
```

make run test2.pla

make run test3.pla