Introduction to Computer I

Final Project

SAT

Instructor: Andy, Yu-Guang Chen Ph.D.

TA: 李孟軒

Department/Class: Electrical Engineering 4A

Name: 陳緯亭

Student ID Number: **109501201**

Contents

1. Pseudo Code

Algorithm 1 DPLL Algorithm

```
1: function DPLL(f: Formula, txtFile: char*): int
2:     result = UNIT_PROPAGATE(f)
3:     \\ result process                                   ▷ stopping conditions
4:     result = PURE_LITERAL_ASSIGN(f)
5:     \\ result process                                   ▷ stopping conditions
6:     i ← DLIS(f)                                         ▷ choose literal
7:     for j ← 0 to 1 do
8:         new_f ← CLONE(f)
9:         new_f.literals[i] ← (new_f.literal_polarity[i] > 0) ? j : (j + 1)%2   ▷ To be 0 or 1
10:        new_f.literal_frequency[i] ← −1                  ▷ already process
11:        result = PURE_LITERAL_ASSIGN(new_f, i)
12:        \\ result process                                ▷ stopping condition
13:        if DPLL(new_f, txtFile) == satisfied then        ▷ recursive
14:            return satisfied
15:        end if
16:    end for
17:    return normal
18: end function
```

Algorithm 2 DLIS Algorithm

```
1: function DLIS(f: Formula)
2:     maxLoc ← 0
3:     max ← 0
4:     for i ← 1 to length(f.literal_frequency) − 1 do
5:         if f.literal_frequency[i] > max then
6:             maxLoc ← i
7:             max ← f.literal_frequency[i]
8:         end if
9:     end for
10:    return maxLoc
11: end function
```

Algorithm 3 Unit Propagate Algorithm

```
1: function UNIT_PROPAGATE(f: Formula)
2:     unit_clause_found ← false
3:     if length(f.clauses) = 0 then
4:         return satisfied                                ▷ empty clauses
```

```
    end if
    repeat
        unit_clause_found ← false
        for i ← 0 to length(f.clauses) − 1 do
            if length(f.clauses[i]) = 1 then                ▷ only one unassigned literal
                unit_clause_found ← true
                \\ Set literal and apply transform          ▷ 1 is neg, 0 is pos
                if result = satisfied or result = unsatisfied then
                    return result
                end if
                break
            else if length(f.clauses[i]) = 0 then           ▷ contains an empty clause
                return unsatisfied
            end if
        end for
    until not unit_clause_found
    return normal
end function
```

Algorithm 4 Pure Literal Assign Algorithm

```
function PURE_LITERAL_ASSIGN(f: Formula)
    unit_pure_found ← false
    repeat
        unit_pure_found ← false
        for i ← 1 to length(f.literal_frequency) − 1 do
            if f.literal_frequency[i] ≠ −1 then
                pos ← (float(f.literal_frequency[i]) + f.literal_polarity[i]) / 2
                neg ← (float(f.literal_frequency[i]) − f.literal_polarity[i]) / 2
                if pos = 0 or neg = 0 then
                    unit_pure_found ← true
                    \\ Set literal and apply transform      ▷ pos=0 set 1, neg = 0 set 0
                    if result = satisfied or result = unsatisfied then
                        return result
                    end if
                end if
            end if
        end for
    until not unit_pure_found
    return normal
end function
```

Algorithm 5 Apply Transform Algorithm

```
1: function APPLY_TRANSFORM(f: Formula, literal_to_apply: long long int)
2:     φ ← A set of clauses
3:     value_to_apply ← f.literals[literal_to_apply]
4:     i ← 0
5:     while i < length(f.clauses) do
6:         j ← 0
7:         while j < length(f.clauses[i]) do
8:             if literal_to_apply × (−1)^value_to_apply = f.clauses[i][j] then
9:                 \\ delete i-st clause
10:                i ← i − 1
11:                if φ is empty then
12:                    return satisfied
13:                end if
14:                break
15:            else if literal_to_apply × (−1)^(value_to_apply+1) = f.clauses[i][j] then
16:                \\ delete the j-st literal in i-st clause
17:                if φ contains an empty clause then
18:                    return unsatisfied
19:                end if
20:                break
21:            else
22:                j ← j + 1
23:            end if
24:        end while
25:        i ← i + 1
26:    end while
27:    return normal
28: end function
```

2. How I compile and execute the program

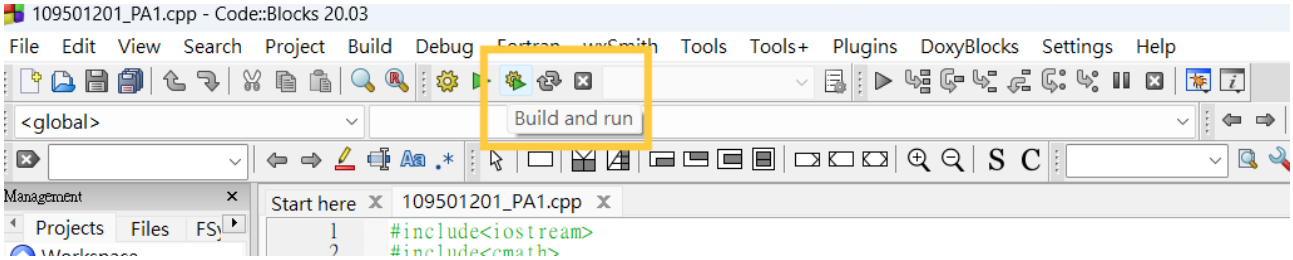

Fig 1: Build and Run

3. PA

3.1 The degree of completion of the assignment: ALL

3.2 Code Explanation

Listing 1: Preprocessor

```
1 #include <iostream>
2 #include <string>
3 #include <vector>
4 #include <fstream>
5 #include <math.h>
```

Listing 2 ，顯示有用到的 Function prototype 和 Struct。

Listing 2: Struct and Function prototype

```
6  enum State { satisfied, unsatisfied, normal};
7
8  struct Formula {
9      vector<long long int> literals, literal_frequency, literal_polarity;
10     vector<vector<long long int>> clauses;
11 };
12
13 void input_file(Formula &f, long long int &numLiteral, long long int &numClause, char
       *cnfFile);
14 int apply_transform(Formula &f, long long int literal_to_apply);
15 int unit_propagate(Formula &f);
16 int pure_literal_assign(Formula &f);
17 void output_file(Formula &f, int result, char *txtFile);
18 int DLIS(Formula f);
19 int DPLL(Formula f, char *txtFile);
```

Listing 3，主要分為讀取 cnf 檔和呼叫的 DPLL 跑遞迴。當 DPLL 這個 function 回傳值為 normal，表示 normal exit program，也就是不滿足，此時要輸出 UNSATISFIABLE。

Listing 3: The main function

```
20 int main(int argc, char *argv[]) {
21     Formula f;
22     long long int numLiteral, numClause;
23     input_file(f, numLiteral, numClause, argv[1]);
24     if (DPLL(f, argv[2]) == normal) {
25         output_file(f, unsatisfied, argv[2]);
26     }
27     return 0;
28 }
```

Listing 4，cnf 檔讀取處理，輸入資料開頭為 p 時，要把 cnf 過濾掉，所以還要用 inFile >> s。接下來，把變數的大小和 clause 的數量分別用變數 numLiteral 和 numClause 記住。clauses 是二維的 vector，去如實記錄 cnf 檔輸入的數值 (含 + 或 - 的資訊)。literal_frequency 和 literal_polarity 則是取 literal 的絕對值進行 vector 定位，分別去紀錄不管該數值正負總出現頻率，以及正和負出現次數相減，該 polarity 可以想成正和負誰出現的比較多，如果 polarity > 0 表示正值出現次數多；反之 prolarity < 0 表示負值出現的比較多。

Listing 4: Input file process

```
29 void input_file(Formula &f, long long int &numLiteral, long long int &numClause, char
       *cnfFile) {
30     ifstream inFile(cnfFile);
31     char c;
32     string s;
33     while (true) {
34         inFile >> c;
35         if (c == 'p') {
36             inFile >> s;
37             break;
38         } else {
39              getline(inFile, s);
40         }
41     }
42     // the number of variable and the number of clauses.
43     inFile >> numLiteral >> numClause;
44     f.literals.resize(numLiteral+1, -1);
45     f.clauses.resize(numClause);
46     f.literal_frequency.resize(numLiteral+1, 0);
47     f.literal_polarity.resize(numLiteral+1, 0);
```

```
48      long long int literal;
49      for (long long int i = 0; i < numClause; i++) {
50          while (1) {
51              inFile >> literal;
52              if(literal == 0){
53                  break;
54              }
55              f.clauses[i].push_back(literal); // put into clauses literal (include
                    information + and -)
56              if (literal > 0) {
57                  f.literal_frequency[literal]++;
58                  f.literal_polarity[literal]++;
59              } else if (literal < 0) {
60                  f.literal_frequency[-literal]++;
61                  f.literal_polarity[-literal]--;
62              }
63          }
64      }
65  }
```

Listing 5，這個 Function 主要配合 unit_propagate 和 pure_literal_assign 做使用。只要指定的值已經被賦予為 literals[literal_to_apply] = 1 (negitive) 或 0 (positive)，就會做相對應的處理，與之同號的，消除其整個 clause；反之與之異號的，刪除其在各個 clause 裡面有出現的該值。在 transform clauses 同時，檢查 clauses 是否全部被消除，如果消除的話，更改整體狀態為 satisfied。如果只有單一 clauses 大小為 0，要更改整體狀態為 unsatisfied，畢竟有還有些 clauses 不為 1。

Listing 5: Apply to transform and simplify the clauses

```
66  int apply_transform(Formula &f, long long int literal_to_apply) {
67      int value_to_apply = f.literals[literal_to_apply];
68      int i = 0;
69      while(i < f.clauses.size()) {
70          int j = 0;
71          while (j < f.clauses[i].size()) {
72              if (literal_to_apply * pow(-1,value_to_apply) == f.clauses[i][j]) {
73                  f.clauses.erase(f.clauses.begin() + i);
74                  i--;
75                  if (f.clauses.size() == 0) {
76                      return satisfied;
77                  }
78                  break;
79              } else if (literal_to_apply * pow(-1,value_to_apply+1) == f.clauses[i][j])
                     {
80                  f.clauses[i].erase(f.clauses[i].begin() + j);
81                  if (f.clauses[i].size() == 0) {
82                      return unsatisfied;
83                  }
```

```
            }else{
                j++;
            }
        }
        i++;
    }
    return normal;
}
```

Listing 6，定義為 When a clause contains only one unassigned literal L, that L must be set to TRUE to satisfy the clause。一個一個 clause 內，搜尋有沒有一個 clause 只有一個 Literal，如果有的話，直接賦予 literals[pos_clause] 其值，賦予的值根據數字其正或負來取決。要注意把 literal_frequency[pos_clause] 設定為 -1，已表示該 literal 已經被賦予值 (1 = False, 0 = True)。賦予值完後，要呼叫先前解釋過的 Listing 5 的 Function - apply_transform，簡化方程式，如果簡化結果為 satisfied 和 unsatisfied，就可以直接 return ，由於該 Function 把 clauses 進行大整頓，所以直接跳出 for 迴圈，重頭開始檢查 unit_clause 的存在。在尋找過程，如果發現只有單一 clause ，表示該方程式還是 unsatisfied，需繼續進行遞迴尋找可行解。

Listing 6: Unit propagate

```
int unit_propagate(Formula &f) {
    bool unit_clause_found = false;
    if (f.clauses.empty()) {
        return satisfied;  // empty clauses
    }
    do {
        unit_clause_found = false;
        for (int i = 0; i < f.clauses.size(); i++) {
            if (f.clauses[i].size() == 1) { // only one unassigned literal
                unit_clause_found = true;
                int pos_clause = (f.clauses[i][0] > 0) ? (f.clauses[i][0]) : (-f.
                    clauses[i][0]);
                f.literals[pos_clause] = (f.clauses[i][0] < 0) ? 1 : 0; // 1 is neg, 0
                     is pos
                f.literal_frequency[pos_clause] = -1;         // already process
                int result = apply_transform(f, pos_clause);
                if (result == satisfied || result == unsatisfied) {
                    return result;
                }
                break;
            } else if (f.clauses[i].size() == 0) { // contains an empty clause
                return unsatisfied;
            }
        }
```

```
114     } while (unit_clause_found);
115     return normal;
116 }
```

Listing 7，定義為 When a literal appears only as either positive or negative throughout the entire formula, without the presence of its opposite form, that literal can be assigned a value to satisfy the entire formula. 搜尋每個 Literal 的，還沒有處理過的 Literal，看 L 和 (¬L) 有沒有只出現其中之一的，如果有的話，直接賦予其數值。等同於看 L 和 (¬L) 是否為 0，用 pos 和 neg 兩個變數分別去紀錄正號出現的頻率和負號出現的頻率。如果正號出現頻率為 0 就表示只有負號出現，就賦予值 literals[i] = 1；反之負號出現頻率為 0 就表示只有正號，需給予 literals[i] = 0。賦予值完後，要呼叫先前解釋過的 Listing 5 的 Function - apply_transform，簡化方程式，如果簡化結果為 satisfied 和 unsatisfied，就可以直接 return 。

Listing 7: Pure literal assign

```
117 int pure_literal_assign(Formula &f){
118     bool unit_pure_found = false;
119     do{
120         unit_pure_found = false;
121         for (int i = 1; i < f.literal_frequency.size(); i++){
122             if(f.literal_frequency[i] != -1){
123                 float pos = (float(f.literal_frequency[i]) + float(f.literal_polarity[
                        i])) / 2;
124                 float neg = (float(f.literal_frequency[i]) - float(f.literal_polarity[
                        i])) / 2;
125                 if(pos == 0 || neg == 0){
126                     unit_pure_found = true;
127                     f.literals[i] = (pos == 0)? 1 : 0;
128                     f.literal_frequency[i] = -1;
129                     int result = apply_transform(f, i);
130                     if (result == satisfied || result == unsatisfied) {
131                         return result;
132                     }
133                 }
134             }
135         }
136     } while (unit_pure_found);
137     return normal;
138 }
```

Listing 8，做輸出 txt 檔處理，當結果是 satisfied ，開始遍歷剛剛決定好的 literals[i] 去了解其為 L 還是 (¬L)。如果 literals[i] 是 1，表示為 (¬L) ，最終要輸出 False；反之，literals[i] 是 0，表示為 L ，最終輸出 True。對於 literals[i]

= -1 的部分輸出 True 或 False 都可以 (我寫 True)。當結果不是 satisfied，輸出 UNSATISFIABLE。

Listing 8: Output file process

```
139 // Dynamic Largest Individual Sum - return Location of the max frequency
140 int DLIS(Formula f) {
141     long long int maxLoc = 0;
142     long long int max = 0;
143     for (long long int i = 1; i < f.literal_frequency.size(); i++) {
144         if (f.literal_frequency[i] > max) {
145             maxLoc = i;
146             max = f.literal_frequency[i];
147         }
148     }
149     return maxLoc;
150 }
```

Listing 9，在 choose literal 的部分，為了不要盲選，先把最大 frequency 的值給選好。在這個 Function ，會回傳最大 frequency 的數值。

Listing 9: Dynamic Largest Individual Sum

```
151 // Dynamic Largest Individual Sum - return Location of the max frequency
152 int DLIS(Formula f) {
153     long long int maxLoc = 0;
154     long long int max = 0;
155     for (long long int i = 1; i < f.literal_frequency.size(); i++) {
156         if (f.literal_frequency[i] > max) {
157             maxLoc = i;
158             max = f.literal_frequency[i];
159         }
160     }
161     return maxLoc;
162 }
```

Listing 10，先進行 unit_propagte，判斷 stopping condition，再進行 pure literal assign，判斷 stopping condition。這樣子可以大大減化 clauses 數量，以及其內部 literals 數。無法被減化的部分，交給 DLIS 選取數值，去分配 literals[i] 數值，看有沒有 satisfied，有 satisfied 就輸出即完成，如果 unsatisfied，就必須繼續分配數值。不斷遞迴 DPLL 去檢查 satisfied。

Listing 10: Davis-Putnam-Logemann-Loveland

```
163 int DPLL(Formula f, char *txtFile) {
164
165     int result = unit_propagate(f);
```

```
166
167     if (result == satisfied) {
168         output_file(f, result, txtFile);
169         return satisfied;
170     } else if (result == unsatisfied) {
171         return normal;
172     }
173
174     result = pure_literal_assign(f);
175
176     if (result == satisfied) {
177         output_file(f, result, txtFile);
178         return satisfied;
179     } else if (result == unsatisfied) {
180         return normal;
181     }
182
183     int i = DLIS(f);
184     for (int j = 0; j < 2; j++) {
185         Formula new_f = f;
186
187         new_f.literals[i] = (new_f.literal_polarity[i] > 0) ? j : (j + 1) % 2; // get
                through DLIS to choose max{freq_x, freq_xbar}
188         new_f.literal_frequency[i] = -1;
189
190         result = apply_transform(new_f, i);
191         if (result == satisfied) {
192             output_file(new_f, result, txtFile);
193             return satisfied;
194         } else if (result == unsatisfied) {
195             continue;
196         }
197         if (DPLL(new_f, txtFile) == satisfied) {
198             return satisfied;
199         }
200     }
201     return normal;
202 }
```

4. The hardness of this assignment / I overcome it

1. 解決遞迴問題。

   Ans: 一開始想遞迴要定義幾種狀態，才能解決這個問題。一開始原本只有設想兩種狀態 satisfied 和 unsatisfied，後面發現這樣子遞迴無法終止。需要有一個回傳 normal 表程式正在正常運作，且還沒判斷完成，才能保證程式正常運轉，且不會無緣無故輸出 UNSATISFIABLE。。

2. 簡化 clauses 增加運算速度的寫法。
   Ans: 一開始原本想使用二維 array 去撰寫 clauses 存取。後面想想，在 unit propagate 和 pure literal assign 的部分會需要減化 clauses，使得運算更快速，那這樣子二維 vector 似乎是更好的容器，有 erase 函數可以運用，就不用刪個東西整串移來移去。

5. Bonus function(s)

5.1 The degree of completion of the Bonus: NONE

6. Suggestions

網路上很多參考資料，認真閱讀做出來很有成就感!