Digital System Design Project 3 – State Minimization

姓名：張睿麟
學號：B11015030
系級：四資工二 乙

先備知識：

1. Sequential Logic [1]
2. State-Transition-Table [2]
3. State Diagram [3]
4. Kiss format [4]
5. Drawing graphs with dot [5]

實作步驟：

1. Build implication table for state pairs.
   建立二維矩陣。
2. Remove output incompatible pairs.
   快篩：對每一格檢查兩個 state 的 input/output 會不會一樣，如果不一樣，就代表此兩個 state 不可能相容，若 input/output 一樣，就代表此兩個 state「有可能」可以化簡。還是需要之後進一步檢查。
3. List transition (next state) pairs.
   找出 current state 他們的 next state 的關係。
4. Check compatibility of transition pair.
   從每一格中，查看是否不相容，不相容的條件是 next state 的對應關係是否剛好為已確定為不相容的 state。
5. Remove incompatible transition pairs.
   每次找到不相容後，要再重投檢查一遍，直至完全沒有。
6. Merge remaining compatible states.
   合併可以相容的兩個狀態，保留第一個出現的，另一個為要刪除的 state。
   要篩除的 state，必須把在表格中有出現的地方，取代成要保留的 state。
7. Update implication table
   在矩陣裡，也要更新，把刪掉的 state 所包含的 row 和 column 都標記

程式說明：

1. 儲存資料的程式

```
struct Next {
    string next;
    string output;
};
struct Pair {
    string first;
    string second;
};
// currentState, binaryInput
map<string, vector<Next> > stateTable;
vector<vector<vector<Pair> > > implicationTable;
vector<vector<bool> > incompatible; // 1 means incompatible.
map<string, int> stateIndex;
vector<string> stateName;
vector<string> deletedStates;
```

2. 化簡的演算法

```
void sieveIncompatible(int inputNum, int stateNum);
void listTransitionPair(int stateNum);
void checkTransitionPair(int stateNum, int inputNum);
void mergeCompatibleState(int stateNum, int inputNum);
```

3. 寫檔案的函式

```
void writeKiss(ofstream& kissOutputFile, int inputNum, int outputNum,
               int termNum, int stateNum, string resetState);
void writeDot(ofstream& dotOutputFile, int inputNum, int outputNum,
              int termNum, int stateNum, string resetState);
void writeInputDot(ofstream& dotInputFile, int inputNum, int outputNum,
                   int termNum, int stateNum, string resetState);
```

4. 輔助函式

```
string intToBinary(int x, int length);
```

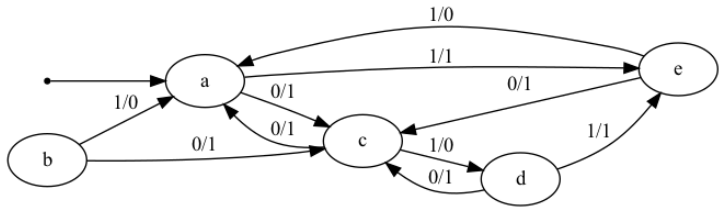
測資演示：

input.kiss	output.kiss
.start_kiss .i 1 .o 1 .p 10 .s 5 .r a 0 a c 1 1 a e 1 0 b c 1 1 b a 0 0 c a 1 1 c d 0 0 d c 1 1 d e 1 0 e c 1 1 e a 0 .end_kiss	.start_kiss .i 1 .o 1 .p 6 .s 3 .r a 0 a c 1 1 a b 1 0 b c 1 1 b a 0 0 c a 1 1 c a 0 .end_kiss

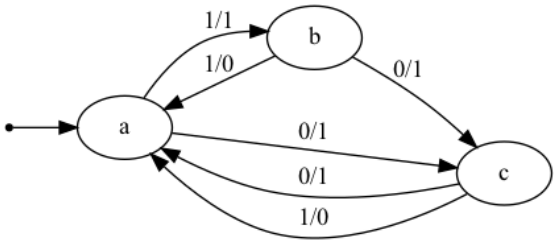
output.dot

```
digraph STG {
  rankdir=LR;

  INIT [shape=point];
  a [label="a"];
  b [label="b"];
  c [label="c"];

  INIT -> a;
  a -> c [label="0/1"];
  a -> b [label="1/1"];
  b -> c [label="0/1"];
  b -> a [label="1/0"];
  c -> a [label="0/1"];
  c -> a [label="1/0"];
}
```

inptut.png



Output.png



参考資料：

[1]: https://en.wikipedia.org/wiki/Sequential_logic

[2]: https://en.wikipedia.org/wiki/State-transition_table

[3]: https://en.wikipedia.org/wiki/State_diagram

[4]: https://course.ece.cmu.edu/~ee760/760docs/blif.pdf

[5]: https://www.graphviz.org/pdf/dotguide.pdf