Dsnp Final Project

FRAIG

(Functionally Reduced And-Inverter Graph)

B07901010
電機二 范詠為
contact inf. 0928900827

1. design of the data structure
   a. strash---the unordered_map

      在 cirstrash 中，為了判斷兩 gate 是否有一模一樣的 input，但又不想用 n^2 的時間複雜度爆搜（兩個兩個都比過一次），而且順序不能亂掉，於是用了 STL 的 unordered_map 去做 hash，而這個 map 的 key 要能讓他分辨出 input 是否一致，所以我先做一次自己定義的 hash 如下（上網查的，return 前先做排序是因為 input 正反是沒差的）：

```
unsigned long long
CirMgr::hashFunction(size_t f1, size_t f2, bool i1, bool i2){
  if(i1 == 1) f1 = 2 * f1 + 0X1;
  if(i2 == 1) f2 = 2 * f2 + 0X1;
  if(f1 > f2) return (f2 << 32) | f1;
  else return (f1 << 32) | f2;
}
```

   b. class FecGrp

      實作 FEC 的部分，我是用 vector _FecGrps<FecGrp*>，也就是分很多組的感覺，用指標可以節省記憶體空間。每一個 FecGrp*指到一個 FecGrp，也就是下面這個 class，主要含有一個 map，這個 map 有點類似_gateList 的 map（<gateid, CirGate*>）但多了最後那個布林值，為的是要判斷反向（相反 simValue 的會在同一組）。另外，在 CirGate.cpp 中，有新加一個 data_member FecGrp*，目的在讓每個 gate 記住他所屬的 FecGrp，在 implement cirgate 指令時會用得到。

```
class FecGrp{
   friend class CirMgr;
   friend class CirGate;
public:
   FecGrp() {}
   FecGrp(unsigned n): _grpName(n) {}
   ~FecGrp() {}

private:
   unsigned _grpName;
   map<unsigned, pair<CirGate*, bool>> _fecList;
};
```

2. algorithms
   a. cirsweep

為了要知道誰是不在_dfsList 中需要被移除的 gate，且不想用 n^2 時間複雜度去找，於是只好在每一個 gate 中記一個_toDel = true，在跑一遍 _dfsList 的過程中，有跑到的改成 false 代表之後要把它移除，接著跑一次 _gateList
做移除的動作。

b. ciroptimize

_dfsList 中是 AIG_GATE 的要去檢查是否那四種情況，再重接，最後做移除，沒什麼特別的做法便不再贅述。

c. cirstrash

在 1-a 其實大概講過了一遍，同樣也是搜_dfsList，如果即將新增的 key 在 unordered_map 有找到的話就表示找到同樣 input 的兩個 gate，接下來就要進行 merge，再刪除，merge 的作法也跟 opt 類似。

d. cirsimulate

一開始先從 FileSim() 做起因為較好跟 ref 比對，而且做好的話 random 應該也不難。我的作法原本是全部的 pattern 都 cin 進來，再 64 個 64 個去做模擬，但後來因為使用的記憶體是 ref 的五倍，所以改成一次 cin 64 個，不但節省了記憶體空間，也增加了一點速度，這部分後面會討論。每一次的模擬分成三個部分，simulate(patterns), genFec(), genLog(patterns, num) 三個部分，simulate 就是給每個 PI_GATE 剛吃進去的 input size_t，從 PO_GATE 跑過 recursive 後更新每個 gate 的 simvalue 值。genLog 則是 outputfile 的部分。

最後最花時間的就是 genFec 了，首先要開始想 FecGrp 的資料型態(1-b)，決定好後實作才不至於遇到太多問題。

接下來我的作法是先 intialize 第 0 個 FecGrp 的東西，再從_FecGrps 開始跑起，每一個進去後要開一個 map<simValue, FecGrp*> newFecGrps，有點似 hash 的感覺，再來先判斷其他的 simvalue 是否跟 base（每個 grp 中的第一個，也就是一定不會被移出 grp 的元素）一樣，一樣的話就不用管去看下一個（當然一樣是指正向反向都不同），不一樣的話就去搜 newFecGrps 看有沒有存在，有的話就根據 newFecGrps 的 second 搬到剛剛有人先搬去的新家，沒有的話就要新開一個 FecGrp*存進去，並 insert 他進去 newFecGrps。

過程中搬家的人最後都要記得自己新的 FecGrp*並把舊的 FecGrp 內的自己刪掉。結束後再用 newFecGrps 的 second 把 size>1 的加入_FecGrps 中，完成 genFec()的過程。

e. cirfraig

fraig 很可惜的是我沒完整做完的部分，我試過跑_FecGrps 跟_dfsList，前者是可以成功的但是因為不是從 input 端證出來每個 satprove 要跑很久，所以 sim13.aag 大概要跑幾十分鐘。但跑_dfsList 的話雖然速度很快，但我 simulate 到最後會進入無窮回圈，用了 cout 大法後發現是跑到後面有可能發生進去_dfsList 之內的每個 gate 存的 FecGrp* _fecgrp 都是 0。想了很久我推測應該是在 merge 的過程中砍掉的 gate 的 fanincone 有一個 grp 的成員都在裡面，造成其實已經結束了但_FecGrps.size()卻不是零的狀況。因為真的沒時間了不然其實應該是想得到解決方法的。

至於我的實作方法是先設定完 satsolver 後，最外層的迴圈就是 if(_FecGrps.size()不是 0)，也就是裡面還有東西的話就要繼續 sat、merge、sim。然後每個 grp（從_dfsList 的 gate 中進去的）都拿第一個(base)跟其他去比，如果比完這個 grp 內都是 UNSAT 的話就代表可以進行 merge，接著因為 merge 後_dfsList 會變，我會跳出去 genDfsList()再進入迴圈繼續跑。而如果不是每個都 UNSAT，就把 SAT 的反例收集起來，每 64 個就 simulate()一次，同時 genFec()，就會有新的 FecGrp*產生，而且會分得更開而有機會裡面再次 prove 會全部 UNSAT 就進行 merge 這樣一直下去直到沒有 grp 了。最後把 initFec 設成 false，這樣接著 cirsimulate 之後就會重新初始化。

3. results and analysis

因為 fraig 沒有很完整只能跑 sim01~sim06 的測資，這邊就放 cirsimulate 在 sim13.aag 跟 ref 的效能比較結果，左邊是 mine，右邊是 ref。