DSNP Final Project

Author

B04611017 林弘曄

Link

https://github.com/linnil1/106_1_DSNP (https://github.com/linnil1/106_1_DSNP)
https://hackmd.io/OwVgbARmCmAMwFoAmYBMSEBYCGAzCCAHAIyybKoDMsN1YYshQA==# (https://hackmd.io/OwVgbARmCmAMwFoAmYBMSEBYCGAzCCAHAIyybKoDMsN1YYshQA==#)

Class

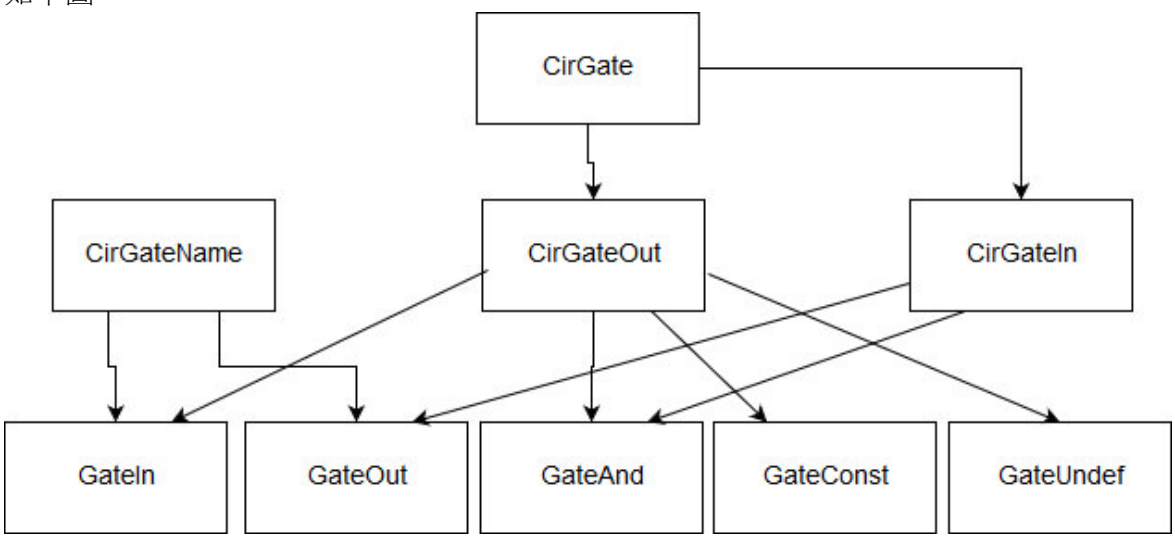
先定義 class

總共有 5 個 class

- GateUndef
- GateConst
- GateIn
- GateOut
- GateAnd

Inheritance

如同拼裝車一樣，他們是的parent class 是 reused 的
如下圖



Gate Design for DFS search

在CirGate中 有這一段程式碼

```
mutable unsigned _visited;
static unsigned  _max_level;
static unsigned  _visited_flag;```
static void setVisitFlag() { ++_visited_flag; }
bool isVisit() const
{
  if (_visited_flag == _visited)
    return true;
  _visited = _visited_flag;
  return false;
}
```

好處是
只有用 `setVisitFlag()` 就可以初始化
然後 call `isVisit()`
第一次 `return False` 代表沒走過
第二次之後 `return True` 代表有走過
這樣的OverHead會全部清掉還小

GateAnd counting

`GateAnd` 中 放一個 static 的 number
construct時就+1， destruct 就-1
不用特定去維護

The size of Fanin

Fanin 可以很省
在 `CirGate` 中使用的是
`getFanin(), getFanin() + getFanSize()`
如果是 一個fanin時

```
ID* getFanin() { return &data; }
unsigned getFanSize() { return 1; }
ID data;
```

如果是 2個fanin時

```
ID* getFanin() { return data; }
unsigned getFanSize() { return 2; }
ID data[2];
```

如此可省空間

Store data type

我define 兩個 unsigned data
`ID` 一個是儲存gate的index，
`DID` (double ID) 是貯存 `ID * 2 + inverted`
反正需要index時 就 `/ 2` ， 需要 inverted 時 就 `& 1`
這樣一個變數就可以解決很多問題ㄌ

other

剩下的細節不在贅述

Useful function in GateMgr

Delete Gate

Delete gate 濃縮成一個function
通常他沒有 fanout ， 或是fanout 等等也不重要，
所以真正要做的是 把 fanin 中的 fanout 刪除
最後才把這個 deltete

```
void CirMgr::delGate(ID& gid) {
  for (unsigned j=0; j<gate->fanInSize(); ++j) {
    CirGate *gchild = getGate(gate->getFanin()[j] >> 1);
    if (gchild)
      static_cast<CirGateOut*>(gchild)->removeFanout((gate->getIndex() << 1) | (gate->getFanin()[j] &
  }
  delete _gates[gid];
  _gates[gid] = NULL;
}
```

Merge Gate

Merge gate 濃縮成一個function
把 fanin 中的 fanout 刪除
把 fanout 中的 fanin ，做一個update，
注意 這個操作 **inverted** 要很小心
最後才把這個 刪掉

```
void CirMgr::merge(ID from, DID to, bool del) // del = true
{
  CirGate *gateFrom = getGate(from),
          *gateTo   = getGate(to >> 1);
  ID inv = to & 1;
  for (const DID &i: gateFrom->getFanout()) {
    static_cast<CirGateOut*>(gateTo)->setFanout(i ^ inv);
    CirGate* gate = getGate(i >> 1);
    dynamic_cast<CirGateIn*>(gate)->updateFanin(from << 1, to);
  }

  if (del)
    delGate(from);
}
```

Find AIG in dfs order

因為走DFS 不如使用 vector 快
所以走一遍後 用 vector 儲存起來
可以省去麻煩
`void goFindAnd(unsigned, IdList&) const;`
可以用在
`writeAag`
`simulate`
時

Sweep

按照Reference的輸出結果
可以判斷出
是先用DFS從outs走完所有的節點
然後再從最前面的 `_gates` 全部掃過一遍
`isVisit()` `return False` 的 就 Delete 掉

Opt

按照Reference的提示
做四種判斷
然後 merge(ID, DID) 起來

strash

這個用Set 就好了
因為是自己的資料結構

所以 我用指標來當作 Key 就好了

因為同樣的Key 只有一筆
所以只用 implement 一個 methods 就好了

insert

如果裡面沒有 就return NULL
如果有 就 return 原來的值

當 return NULl 時， 我們就放進去不理她
反之 則把 這個節點 跟 return 回來的 Merge

```
HashData* insert(const HashKey& k, const HashData& d) {
  VHashNode &v = _buckets[bucketNum(k)];
  typename VHashNode::iterator it = find_if(v.begin(), v.end(),
    [&](HashNode& p) { return *(p.first) == *k; });
  if (it != v.end())
    return &it->second;
  v.push_back(HashNode(k, d));
  return NULL;
}
```

其中裡面我用Lambda function (for c++11) 來做比較

Compare and hash

因為要沒有順序 a,b = b,a
所以先判斷小的 才乘上 某個值

比較一不一樣時，也是是先判斷小的，然後比較，再比較大的

```
size_t GateAnd::operator () () const {
  bool c = _fanin[0] > _fanin[1];
  return 888777 * _fanin[c] + _fanin[!c];
}

bool GateAnd::operator == (const GateAnd& b) const {
  bool ia =   _fanin[0] >   _fanin[1],
       ib = b._fanin[0] > b._fanin[1];
  return (_fanin[ia] == b._fanin[ib]) && (_fanin[!ia] == b._fanin[!ib]);
}
```

Simulate

在每個Gate 裡面
都放一個 `void simulate()` 在裡面

這樣不用判斷他是哪種Gate 了

然後用 goFindAnd() 把所有要simulate 的 gate 儲存成 vector
因為是 拓樸排序 所以可以直接 simulate

FEC

用map來分類
第一個是 `pair<之前的 group num, 先在模擬完的 group num>`
然後 先算 size
如果 `size > 1`
再用 DID 一個個重新編號組別number
只有全程都是從最小開始做
那就不用 sort 了

```
typedef pair<DID, Value> PV; // orignal group, new group
map<PV, pair<unsigned, DID> > m; // size, groupId
```

不過 map 是 `nlogn` 的算法
所以速度會比較慢一點

```
Your ans : Period time used : 0 seconds

Ref  ans : Period time used : 0.01 seconds

Your ans : Total time used  : 0 seconds

Ref  ans : Total time used  : 0.01 seconds

Your ans : Total time used  : 0.01 seconds

Ref  ans : Total time used  : 0.02 seconds

Your ans : Period time used : 0.05 seconds

Ref  ans : Period time used : 0.06 seconds

Your ans : Total time used  : 0.06 seconds

Ref  ans : Total time used  : 0.08 seconds

Your ans : Period time used : 0.21 seconds

Ref  ans : Period time used : 0.2 seconds

Your ans : Total time used  : 0.27 seconds

Ref  ans : Total time used  : 0.28 seconds
```

不過也沒慢多少

Fraig

把程式碼
strash opt 都複製過來
再加上 SatSovler 每次證明完都不用刪掉
結果都比 ref 快

```
$ python3 my_test_fraig.py
tests.fraig/sim07.aag :
RUN OK
RUN OK
MY
Period time used : 0 seconds
Total time used  : 0 seconds
Total memory used: 0 M Bytes
Period time used : 0.01 seconds
Total time used  : 0.01 seconds
Total memory used: 0 M Bytes
Period time used : 1.61 seconds
Total time used  : 1.62 seconds
Total memory used: 0 M Bytes
----------------------------------------
REF
Period time used : 0 seconds
Total time used  : 0 seconds
Total memory used: 0 M Bytes
Period time used : 0.02 seconds
Total time used  : 0.02 seconds
Total memory used: 0 M Bytes
Period time used : 7.24 seconds
Total time used  : 7.26 seconds
Total memory used: 0 M Bytes
```

順帶一題
為了驗證 flaig 我特別寫了一枝程式 100 行 就搞定ㄌ
在 `src/sat/checker/satTest` 裡