黃彥翔
電機二 B06901170
DSnP Final project REPORT
2019.1.17

Mail : b06901170@ntu.edu.tw
Phone : 0968836103

Design of the data structure :

CirGates member introduction				
CLASS	繼承	member型態	名稱	功能
CirGate	Base	vector <CirGate*>	parent	儲存fanouts
		int	id	儲存gates id
		int	travel	紀錄在dfs時的經過次數
		size_t	sim	儲存simulation 的value
CirAigGate	Derived	CirGate*	Lchild	儲存fanins
			Rchild	
CirPoGate	Derived	CirGate*	child	
CirPiGate	Derived			

function of Gates which are frequently used:

Return type	名稱	功能
Void	merge (CirGate* _merged)	merge”_merged”.同時更新與_merged相關的gates的fanin or fanout
CirGate*	invertprt (CirGate* _gate)	Return _gate 的 inverting 型態。(CirGate*)(++((size_t)_gate))
CirGate*	oriprt (CirGate*)	Return _gate 的 original 型態。

Data structure 簡介：
Gates 互相連接，往上可以接無限個fanouts；往下可以接2個（AIG）或1個（Po）fanin(s)。fanouts 或 fanins 儲存的方法為：若是inverting gates，則儲存(CirGate*)(++((size_t)_gate))；若否，則儲存原先的adress。

members of CirMgr:

CirMgr member introduction		
member型態	名稱	功能
vector <CirGate*>	cirGate	儲存全部的gates
vector <CirGate*>	piList	儲存CirPiGates
vector < vector< int > >	_dfsList	紀錄在dfs時gates的經過順序
vector < vector< size_t > >	used_pattern	儲存fraig後所產生的useful patterns
size_t	po_number	紀錄po數量
size_t	piaig_number	紀錄最初始時的CirPiGates 和CirAigGates數量總和

Algorithms of simulation:

步驟	目的	實現方法
(1)	讀取檔案	以for迴圈一次讀取64個（string) 迴圈裡每一次將CirPiGate::sim 和新讀進來的bit做bitwise or 。
(1)	Random	Random 生成piList.size( )大小的size_t陣列。
2	取得 gates -> sim	以遞迴 dfs 方式呼叫gates 中的simulation function 。 simulation function： 以bool CirAigGate::simulate( )為例，sim 的取得為：先取得Lchild -> sim 和 Rchild -> sim ，再bitwise or 起來 。
(3)	初始化 FecGrps	將所有在_dfslist中的gates push_bck 進FecGrps[0]。
4	fecgrp分類	將每個fecgrp的成員按照其sim value丟進hash進行分類。 若找到一樣的sim value（hash_key），則將此 gates -> id 加進vector（hash_value）； ＊ hash： unordered_map < size_t , vector< int > > fecMap ＊ hash_key : sim value ＊ hash_value : a vector of int(gate's id) .
5	更新 FecGrps	對於每一個fecMap 的member，若其vector（hash_value）size() == 1，則將其拋棄；反之則將此vector（hash_value）push_back進FecGrps。 最後要將原先就再FecGrps中的vector erase掉，避免重複。
6	排序	對於所有在 FecGrps 中的vector，對於其中的id做排序。 之後對於FecGrps，依照其中的vector[0]做排序。
***	加快sim -r 速度	由於每一次的Fraig 完，都會剩下許多 useful pattern，因此我用 CirMgr -> used_pattern 存下來，下一次的操作時，先以這些 pattern simulate。
***	simulation 停止機制	當每50次simulation，FecGrps.size()下降少於5次，或是總共的 simulation pattern數量 == cirGate.size( ) / 50 ，便停止 simulation。

Algorithms of FRAIG:

步驟	目的	實現方法
1	建立 solver	＊須將_const0 接到 任意已知存在的gate。 solver.addAigCNF(_const0 -> _var , v , true , v , false);
2	排序	須保證FecGrps中的任意一個group，其第一個 gate 是整個 FecGrp 中，位於_dfslist最前面的。 故按照_dfslist順序，將gate push_back到group的尾端，如此一來，group內部將按照_dfslist排序。
3	solve SAT	按照dfslist順序，將每一個gate與group中的第一個gate 進行SAT solve。若 equivalent ，則用 “group中的第一個gate” merge 此 gate，如此一來可以避免電路cycle發生。 若 inequivalent ，則收集pattern，且將此pattern更新給每個gates中的sim value 。 之後的gate ，若要進行SAT solve ，可以先檢查這些pattern。
4	去除多餘的gates	呼叫strash
5	重新建立 FecGroups	再次呼叫 random simulation
6	**Repeat 1,2 直到FecGrps.size() < cirGate.size()/n**	
7	solve SAT	＊與step 3.不同，這一次會將所有在fecgroup中的gates組合都證過一次。
8	去除多餘的gates	呼叫strash

Result analysis for experiment:

實驗一：
simulation中，控制fecgroups下降速度對於總體時間、memory、gates 數量的影響。
（n=100）

	對照組 (ref code)	下降速率 ＝15組/50次	下降速率 ＝10組/50次	下降速率 ＝5組/50次	下降速率 ＝5組/100次	下降速率 ＝3組/100次	下降速率 ＝2組/100次
時間	68s	54.44s	45.91s	42.2s	52.22s	44.7s	50.62s
memory	44.52MB	45.14MB	44.06MB	44.73MB	42.71MB	41.71MB	42.52MB
Total Gates	84020	83961	83961	83961	83961	83961	83961

實驗二：
fraig中，控制step6中的n值（FecGrps.size() , cirGate.size() 的比值）對於總體時間、memory、gates 數量的影響。（下降速率＝5組/50次）

	對照組 (ref code)	n=50	n=80	n=100	n=200	n=500	n=1000
時間	68s	48.5s	48.28s	48.98s	46.41s	47.69s	54.88s
memory	44.52MB	44.12MB	44.73MB	40.45MB	45.77MB	43.92MB	41.79MB
Total Gates	84020	83961	83961	83961	83961	83961	83961

結論：
兩組實驗都會有最佳值。
在實驗一中，當simulate太少次，會有太多組fecpairs要證明，因此耗時間；但是若simulate太多次，後面幾次反而沒有太多作用。
在實驗二中，n太小，最後一次的“全部證明的部分”會有太多組fecpairs，因此耗時；但是若n太大，反而會太晚進入全部證明的部分，多太多時間在猜pattern。