資料結構與程式設計

Functionally Reduced And-Inverter Graph (FRAIG)

Final homework

姓名：王佩琳

學號：B06705048

聯絡資訊：

信箱：bluesky5518@gmail.com

FB：王佩琳

1. Design of the Data Structure

1.1 cirGate

- Member variables：

型態	名稱	用途
unsigned	_id	Gate 的 id
GateType	_type	Gate 的 type
int	_line	Gate 生成在第幾行
unsigned	_ref;	DFS 標記用
bool	_inv[2]	紀錄 fanins 的 inverse
CirGate*	_faninList[2]	紀錄 fanins 的 id
GateList	_fanoutList	紀錄 fanouts 的 id
string	_name	紀錄名字
bool	_isUsed	Sweepinp 時判斷有沒有 used
bool	group_exist	FEC 時判斷有沒有已經在某個 group
size_t	last	最後 64 次 patterns 的 simValue

- Member functions：

回傳型態	名稱	用途
string	getTypeStr()	回傳 type
virtual bool	addFanin	加入 fanins
virtual bool	addFanout	加入 fanouts
unsigned	getLineNo()	回傳 line
virtual bool	isAig()	判斷是否為 AIG
unsigned	get_id()	回傳 id
virtual bool	get_inv(int i)	回傳第 i 個 fanin 有無 inverse
void	add_name(const string)	加入 name
static unsigned&	_globalRef()	DFS 用
bool	isGlobalRef()	isMarked
void	setToGlobalRef()	mark
tatic	void setGlobalRef()	clearMark
void	traversal	Recursion DFS
void	reportRecursive	對 gate 的 cirp –fanin i
void	reportRecursive_out	對 gate 的 cirp –faout i
virtual void	printGate()	印出 gate 資訊

void	reportGate()	Report Gate
void	reportFanin	Report Fanin
void	reportFanout	Report Fanout
void	change_inv	更改 fanin 的 inverse

Sub class：

AigGate、PiGate、PoGate、UndefGate ConstGate

1.2 cirMgr

- Member variables：(部分略)

型態	名稱	用途
unsigned	maxnum	Circuit 的最大 index
unsigned	inputs	Circuit 的 PI 數
unsigned	latch	Circuit 的 latch
unsigned	outputs	Circuit 的 PO 數
unsigned	ands	Circuit 的 AIG 數
unsigned	_patterns	已經執行多少次 patterns
vector<vector<int>>	fecGrps	FEC Group
bool	const_exist	是否有 const 0
GateList	_piList	Vector<CirGate*> 記錄所有 pi
GateList	_poList	Vector<CirGate*> 記錄所有 po
CirGate**	_total	記錄所有 gate
vector<size_t>*	_pi	紀錄所有 pi 的 simValue
vector<size_t>*	_all	紀錄所有 gate 的 simValue

- Member functions：(部分略)

回傳型態	名稱	用途
void	Merge(int A, int B)	將 gate B 被 gate A 取代
void	run_simulation()	對一次 pattern 計算 simValue
void	FEC()	對目前的 patterns 結果拆分 fecGrps
void	separate()	對一次 pattern 結果拆分 fecGrps

2. Algorothms

2.1 Sweeping

功能：移除 circuit 中，對 PO gates 為 unreachable 的 gates

作法：稍微修改 dfs() function，傳入的 arguments 增加一個 isSweep 的 bool，如果值為 true，在 traversal 並記錄 dfs list 的同時，將所有 list 上 gate 的 isUsed 值更新為 true，隨後檢查所有 gates 的 isUsed，將所有 isUsed 為 false 的 AIG gate 進行移除，移除步驟如下：

1. 對要移除的 gate A 的 fanins 中的 fanoutList 移除 gate A
2. 對要移除的 gate A 的 fanouts 中的 fanins 移除 gate A
3. 移除 gate A
4. 修正當前 AIG 數

原理：對於每次讀入 circuit 後，並沒有在 read 時就建立 dfs list 存在 cirMg 中，考慮到後續如果執行包含移除或取代功能的指令，需要不停更新已經記錄下來的 dfs list，作法過於繁複，因而選擇在每個功能有需要時呼叫 dfs()，直接在 traversal 後對 circuit 動作，在 cirp –ne 的時候，再跑一次 dfs()，印出來的即為最後更新的 circuit

問題：如果一個 AIG 是 unreachable 的，代表它的 fanout，它 fanout 的 fanout 都是 unreachable，直觀來講上述做法沒有問題，但是否過於冗，另一想法是，對每個要移除的 AIG 檢查它的 fanin，如果為 PI，將 PI 的 fanout 移除此 AIG，再直接移除此 AIG，但又不確定會不會出現記憶體已歸還仍試圖存取的問題，如：AIG destructor 刪掉了 fanout，後面又想存取它的 fanout 去進行下一個 AIG 的移除。

2.2 Optimization

功能：對於四種 case 進行 optimize

(a) fanin 有 const 1，AIG 被取代為另一 fanin

(b) fanin 有 const 0，AIG 被取代為 const 0

(c) fanins 相同，AIG 被取代為它的 fanin

(d)fanins 為 inverse，AIG 被取代為 const 0

做法：先執行 dfs()，對每個 dfs list 上的 AIG，觀察它的 fanin 的 id 和 inverse，處理*2+1 後判斷是否有出現上述 case，如果有，執行以下步驟：

令：AIG A 為要取代成的 gate，AIG B 為要被取代的 gate

1. 對 B 的 fanin 的 fanout 移除 B，
2. 對 B 的 fanout 的 fanin 移除 B，加上 A
3. 對 A 的 fanout 加上 B 的 fanout
4. 移除 B，修正當前 AIG 數

原理：對於出現上述 case 的 gate 找到即將取代和被取代的 gate A 、B，修正 optimize 後的串接。需要額外注意，在 case(a)和 case(c)要更新原先 B 的 fanout 存取的 inverse，如：

8 7 7

12 8 10

Simplifying: 3 merging !4

要更新 6 的_inv(0)為 true

問題 1：考慮到可能原本 A 的 fanout 就存在 B 的 fanout，在步驟 4 時有無需要加上 iterator 尋找，避免 A 的 fanout 出現重複的 gate，如：

16 14 14

18 14 16

Gate 8 因 case c 被 Gate 7 取代。

雖然 vector 的 find 時間複雜度只有 log(n)，但多數應該不會出現這個情況，每次都要先檢查過於麻煩；但如果不先檢查直接加入造成重複，在跑其它的功能時存取到這個 gate 的 fanout 時，怕會出現錯誤。

問題 2：inverse 是被記錄在 AIG 或 PO 裡，一個 gate 像是如下圖(a)被記錄，但

在 optimize 時考慮的 replace 卻像是圖(b)，會影響到下一個 gate 的 inverse，很容易造成更改錯誤，考慮更改 inverse 在 gate 中的紀錄方式，但以 circuit 串接來說並不合理，又考慮或者一開始在記錄 fanin id 時保留*2+1 的處理，但此作法又會造成其他時間取出和判斷時的麻煩。

圖(a)　　圖(b)

2.3 Strash

功能：**merge structurally equivalent 的 signals**

做法：先執行 dfs()紀錄 list，然後每次對在 dfs list 上的 AIG，將它的 fanins 處理後(*2+1) 組成 OnePart， 此為有兩個為 int member 的 class，生成 object 時會將較小的 fanin id 存在其中的 one，以 OnePart 為 keyValue 在 HashMap 查找，如果有找到，執行 merge()，如果沒找到，以 <OnePart, int>放入 HashMap，直到走遍所有在 dfs list 上的 AIG。

原理：比較後對 fanin 一樣的 gate 做 merge()：

merge()：

令：AIG A 為要取代成的 gate，AIG B 為要被取代的 gate

1. 對 B 的 fanin 的 fanout 移除 B，
2. 對 B 的 fanout 的 fanin 移除 B，加上 A
3. 對 A 的 fanout 加上 B 的 fanout
4. 移除 B，修正當前 AIG 數

其中，步驟 1 中 B 的 fanin 的 fanout 不用加上 A，因為本來就存在了。

問題：optimize 和 strash 都是一個 gate 取代另一個 gate，應該要有一個 replace 的 function 能傳入：要取代成的 gate A 和要被取代的 gate B 去做 update circuit，而非分開來寫，code 相似過於冗長且維護不易。

2.4 Simulation

功能：對 circuit 做 boolean logic simulation

作法：

1. randomSim()：

對 circuit 進行 64 * 隨機 round 的 patterns，考慮到所有可能造成出現差異 value 的 patterns 為 2^(PI 的個數)，且 simulation 是為了將 singnals 分開成 FEC，和 AIG 也會有關，去推測一個合適的 round，以 randomSimData()取得隨機的 size_t 放入 cirMgr 的 vector<size_t>* _pi 和_all 裡，呼叫 run_simulation 去計算並記錄，再呼叫 FEC()進行分拆，如果 command 時有輸入-o，即__simLog != NULL，將計算好的結果輸出至指定檔案，包含 pi 和 po 每次 pattern 下的 simValue。

2.fileSim(ifstream&)

從檔案中讀取 PI 的指定 boolean，每超過 64 個 pattern 就先放入為 0 的 size_t 進每一個_pi[]，依據讀取到的 boolean 去 set size_t 的 bit，隨後計算如 randomSim 的後半部分。

3.randomSimData()：

對一個為 0 的 size_t num，隨機生成 0~63 的數字 n，將 num 的第 n 個 bit 設置為 1，重複上述的動作隨機次數，回傳 num

4.run_simulation()

計算並紀錄一次 pattern 下所有 gates 的 SimValue，使用 All-gate simulation，對所有在 dfs list 的 gates 做運算，evaluation 用 if-else 判斷：&&連接 fanins 在正確 bit 位置的值，如果 and 下為 true，將 gate 的 simValue 對應到的位置 set the bit

5.separate()

對單一次的 pattern 判斷是否要分開。對每個在 fecGrps 裡的 fecGrp 中的每個 gate 判斷此次 pattern 下的 SimValue 為何，生成兩個新的 fecGrp 並依值分開放入，移去原 fecGrp，直到對所有原本即在 fecGrps 的 fecGrp 運作完。

6.FEC()

```
typedef vector<vector<unsigned>>   fecGrps;
typedef vector<unsigned>           fecGrp;
```

生成一個 fecGrp 將所有的 singals(AIG 和 const 0)放入，再把這個 fecGrp 放入 fecGrps[0]，對前面重複 run_simulation()跑出來的_all 裡的每次 pattern 的每個 gate 的 simValue 進行 separate()，重複直到所有目前蒐集到的 pattern 結果判斷結束。

原理：用 size_t 以 Parallel pattern 的方式去 simulate gate 的 simValue，計算完後再對會出現不同結果的 gates 拆成不同的 FEC。

問題：沒有看懂老師講議上用 Hash 的方法，使用 vector<vector<unsigned>> fecGrps 記錄所有的 group 的缺點是，在不停的二分下，會需要一直產生新的 vector，而一個 vector 的 size 為 24bytes，非常的占空間，且此存取方式造成一瓶頸為，在 cirp –fec 時，沒辦法依照規定做到每個 group 最前頭為正且依序遞增，每個 group 第一個值也是遞增的。

2.5 fraig

功能：將所有 SAT-proven equivalent 的 gates merge

問題：前一項 sim 功能會分錯 fec group，無法建立在此基礎上進行 SAT-proven

3. Experiments - the results and analysis -

考慮對前 sim01~sim06 分別做 sweep 和 optimize 及 strash，會發現在小 size 的情況下，空間和時間趨近於零，而在 sim06 時 420.3 M Bytes 的慘劇，ref 仍維持約 1M Bytes 的空間。