資料結構與程式設計期末專題

DSnP Final Project

姓名：蘇峯廣

學號：B04901070

指導教授：黃鐘揚教授

繳交日期：2017/01/18

Email：b04901070@gmail.com

資料結構與程式設計期末專題

Functionally Reduced And-Inverter Graph (FRAIG)

零、 大綱

壹、 專題目的

藉由 Unused gate sweeping、Trivial optimization、Simplification by structural hash、Previous Simulation 等前四個方法將電路用有系統且有效率的方法進行電路的初步化簡，再經由呼叫 Boolean Satisfiability (SAT) solver 的方法進行 FRAIG: Equivalence gate merging，進而達到縮小電路面積與邏輯的一套演算法之實行。

貳、 函式的執行方法及優缺點分析

一、 Unused gate sweeping

藉由呼叫 CirMgr::sweep 來實行，每一個 CirGate 中存有一個稱為 bool sweepdfT，負責記錄是否 Depth First Search Traversal 有走到該 CirGate，若沒有走到而且該 CirGate 為 AIG_GATE 的時候則將其刪除，較為特別的地方在於如果 fanin 為 unused gate，則其所有 fanout 皆為 unused gates，所以只需處理好 fanin 的連接即可，這樣大幅減少邏輯的複雜度及運算時間，需要注意的地方基本上只要注意不要刪除到 input 跟 ouput 即可。

二、 Trivial optimization

藉由呼叫 CirMgr::optimize 來實行，總共分成四個個案來討論，第一為當左右都一樣，則將它取代為一個 fanin 即可。第二為當左右一樣但一個反向一個正向，此時則將它取代為 0。第三為當左右有一個為 1 時，則將它取代為非 1 的那個 fanin。最後為當左右有一個是 0 時，則將它取代為 0。藉由這四個個案就可以輕鬆做出 optimization，基本上 optimization 相對上問題少很多，運算時間也只要走過一次 Depth First Search Traversal 即可。

三、 Simplification by structural hash

以下將分成兩個部分做討論：

第一部分為我採用 HashMap 的作法，每一個 HashNode 的第一個元素為 HashKey，負責儲存兩個 fanin 資訊，而我將（）overload 成兩個 fanin 的 ID 加一相乘，這是我到現在將所有的 CirGate 分的最開的方法，重複率極低，而至於第二個元素則為最基本的 CirGate*。至於 HashMap 的實際操作，改用了跟之前不一樣的手法，加速了許多也 de 掉不必要的 bug，我在 iterator 裡不再存著 Hashnode，而是改成用他二維位置的座標來儲存，也解決之前 end()塞了一個 dummy node 衍生的各種問題，而其他的函數基本上都跟之前的作業七是類似的。

第二部分為 CirMgr::strash 的實際實行，則大量運用 HashMap 尋找快速的好處，將整個電路跑過一次 strash，將重複的進行取代，這一部份算是相對簡潔，困難的地方在於一開始 HashMap 的構思，速度上面因為有 HashMap 的關係，運行速度都很快，基本上最多一到兩秒就會執行完畢。

四、 Previous Simulation

以下將分成三個部分做討論：

第一部分為 CirMgr::fileSim，實行方法是將讀進來的 patterns 轉成 32 個一組的 parallel _FECnum 賦予 input，若是 UNDEF 的 CirGate 則將它設置為 0，然後 CirMgr::for_each，相對困難的點在於如何印出格式要求的檔案，所以我採用的是一次印出 32 個 patterns 的資料，但這也有一個缺點是，如果 patterns 的資料不是 32 的倍數的話，結果可能跟 reference 不一樣，但基本上只要 patterns 是 32 的倍數皆可以運行。

第二部分為 CirMgr::randomSim，實行方法則為藉由呼叫 rnGen 的方法，將 input 隨機的賦予值，比較特別的地方是，我連 UNDEF 的部分都賦予了隨機的值，雖然沒有進入 CirMgr::for_each，但我發現這樣可以少掉許多 simulation 後為 0 的 CirGate 數，這一部份比較需要思考的地方應該是 0 的輸入以及輸入為正向或反向的困難度，第一個我將 0 強制輸入的作法，而不在討論時輸入，可以確保 0 必在 FecGrp 的第一位好讓 Fraig 執行之外，也可以確保 0 必在裡面。第二個我將正向的 _FECnum 與反向的_FECnum 一起呈現做尋找，讓第一位為正向，其他與之比較分出正反向，成功解決以上我發現的兩個大問題。

第三部分為 CirMgr::for_each 的實際操作，我開了一個新的 NewFecGrp 儲存新分出的 FecGrp，接著用 CirMgr::ckeck_valid_Grp 來尋找是否有 size 為 1 的 FecGrp，至於 MAX_FAILS 的取法，我選擇用 $3.5^{*}(\text{input})^{2}$，因為我覺得 input 數佔了很大的關係，若 input 多，變化就會多；相反的 input 若少，變化也跟著減少。

中間的操作皆為 HashMap，其中第一個元素為 SimValue 負責記住 simulation 後的值，而第二個元素為 vector<CirGate* >用來儲存相同 SimValue 的所有 CirGate，但問題就發生在 HashMap 身上，雖然 HashMap 尋找較快，但是一旦需要先 query 在 update 的時候，運行速度在這裡慢下來很多，為了解決這個問題，我在這裡用了一點秘訣，就是多一個 bool 值叫做 tricky，當總個數超過 20000 個的時候，他一次只 simulate 五次 fail，這個靈感來自於我對於 reference 的比較發現，運行五次 fail 跟運行 181 次 fail 並沒有差到很多，而先 Fraig 有一個好處可以先大幅降低總個數，此時再回來 simulation 就可以大幅降低運行速度，但缺點是，當總個數真的過大時，需要重複極多次先前的步驟。優缺點參半，但經過多次運行我覺得優點似乎大於了缺點一點點，所以我採取這樣的作法。

結論基本上都可以運行在可接受的速度上，最多幾十秒，除了 sim13 的測資以外幾乎沒有問題。

五、 FRAIG: Equivalence gate merging

藉由呼叫 CirMgr::fraig 來實行，這部分的困難度相對輕微，只需要熟悉 Boolean Satisfiability (SAT) solver 即可，我的實際做法為，將所有的 CirGate 跟第一個做 Fraig，最多做三次，直到次數限制或是已

經全部分開完為止，這時候就用到前面的技巧：

第一是 0 必定在第一個，好處在於因為後面都是用前面的 merge，所以取代的函式相對清晰，而且若取代為 0，電路必定更加好簡化。

第二是若該 CirGate 已經被 Fraig 過，則將它的一個名為 fraiged 的 bool 值設成 true，則再次 simulation 之時，就不會將他在丟入 CirMgr::for_each 之中。

第三為 tricky 的運用，當這個資料量很大而且這一個 FecGrp 的 size 超過 100 個的時候，一次只 fraig 100 個，大量降低所需要的運行速度，但缺點跟之前的類似，就是當總數量真的過大時，將會需要多次的 simulation 加上 fraig。相同的，我選擇這個的原因即為優點似乎比缺點多那麼一點點。

最後為 0 的實行，我採用 assumeProperty 為 false 的方法進行實行。結論上，除了 sim13 所需次數極多之外，其他測資在 Fraig 2 次以內可以結束，那也是我選擇兩次的原因。

參、 運行結果的分析及結論

經由我幾天下來運用所有測資測出的結果來看：

第一部分：Unused gate sweeping、Trivial optimization、Simplification by structural hash 的運行皆十分順暢，幾乎沒有半點缺失的部分，運行速度上最多都是個位數的秒數，這部分應該算是相當成熟了，沒有 bottleneck 問題存在。

第二部分：Previous Simulation 跟 FRAIG: Equivalence gate merging 的部分，除了 sim13 的測資以外，都運行的十分順暢，bottleneck 的問題就發生在 sim13 這種總個數極多的測資身上，加了 tricky 之後，運行變得順暢了，但 bottleneck 的缺點就跟先前討論的一樣，總個數多時需要多次的 simulation 以及 fraig。

肆、 參考資料

由於此專題用到了 minisat 的相關技術，故貼上其參考資料及申明：

the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software isfurnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies orsubstantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.