DSnP Final Project

Fraig

(Functionally Reduced And-Inverter Graph)

B03901030電機三 蕭晨豪

email:b03901030@ntu.edu.tw

2017/1/18

■ Abstract

The final project is an implementation of FRAIG. The parser read the AIGER format and construct the circuit. After that it offers various operations to simplify the circuit. The details of the operations will be discussed later.

■ Data Structure

AIGER format has only and gates and allowed inverting input. So the circuit construction is simple.

● Gate

Use class CirGate to represent the circuit gates. It can be PI, PO, AND gate, Constant 0 and Undefined. For each gate it has vectors of pointer of gate to store its fanins and fanouts. And use the last bit of the pointer to represent the inverting/non-inverting type. Each gate also has a pointer to trace back its FEC group after doing simulation.

● Circuit

Use the class CirMgr (manager) to manage all the operations. The class has vectors for all the PIs, POs, gates...etc. And a list of FECGroup (a class defined to manage the FEC groups) to store all the FEC groups.

■ Algorithm

● Sweep

Simply sweep the gate that PO can’t arrive.

● Optimization

There are four simple cases to simplify the circuit, briefly describe the cases and the algorithm I used:

1. Has a CONST0 input: replace the gate with a CONST0.

2. Has a CONST1 input: discard the gate, connect the non-CONST1 input gate to original gate's fanout properly.

3. Has same input with same inverting state: replace the gate with its input.

4. Has same input with different inverting state: replace the gate with CONST0.

● Structural Hash (Strash)

The purpose is to merge gates that has the same fanins. Design a class HashMap to compare the AIG gate's two fanins. The hash function is designed to return a specific value depends on the gate's fanins.

My choice is to store the gate's (fanin value << 1 | inverting state) as variable _input0 and _input1. The return hash key is a size_t variable :

(smaller input << 32 + larger input)

The 32 bit shift is to prevent the collision of _input0 and _input1.

The restriction that the "smaller input" should be shifted promises that the gate with fanin (a, b) and one with fanin (b, a) would have the same hash key and can be strashed.

Simulation

Purpose is to find FEC groups by feeding random or file-specified inputs.

Use parallel pattern (pack 64 input values into a size_t) to do the simulation.

Each gate has a size_t variable _simvalue to record the current simulation value.

Define a class FECGroup to manage the FEC groups. A list of FECGroups is created in the Circuit manager (CirMgr) to collect the FEC groups.

Use HashMap data structure to collect the gates with same (or exactly inverting) simulation value, so the hash function is simply the current simulation value.

The simulation is done by the following step:

1. Create a HashMap newgrps to store and find the (simulation value, FEC group) pair.

2. If the operation is specified random, set the fail threshold be log2 (# of PIs + # of AIG gates).If cannot find new FEC groups after the simulation, the fail counter ++ . The simulation process halts when the fail count exceeds the threshold.

3. Assign simulation value to all gates in dfs list, and collect the FEC groups with same simulation value.

4. Assign the pointer of FECGroup to all the gates in dfs list. This allows the gates to trace back its FECGroup.

FRAIG

Call minisat engine to formally prove the satisfiablility problem that assigned to the SatSolver object. We use this to prove the equivalence of gates in the same FEC groups. (The engine also generate a counter example if UNSAT)

My fraig operation consists of two parts: Fraig the CONST0 group and fraig all the others.

First we generate the proof model of all gates (by calling addAndCNF() for AIG gates and assumeProperty() for CONST gate).

For CONST0 group we call assumeProperty() for the gate we want to prove and get SAT/UNSAT result. Merge the gates with CONST0 (or left it intact) by the result.

For all the other FEC groups we simply compare the gates with the group's first gate. (Brutal and slow though)And merge the gates (of left it intact) by the result.

※There is a serious integrity problem in Fraig() and I can't fix it before deadline. I delete the (delete CirGate*) code to make my Fraig() work, but after Fraig() some pointer that should be deleted remain intact. This will cause segmentation fault when read in another AIGER file by the

command –replace.

■ Experiments and Analysis

The experiment consist of two parts:

1. Performance

Use command cirsim –r to tests the performance of simulation.

File	My program	Reference program
Sim07 (# of PIs + # of AIGs:9441)	896 patterns simulated cirsim –r: 0.1s	512 patterns simulated cirsim –r: 0.02s
Sim09	1216 patterns simulated	1888 patterns simulated

(# of PIs + # of AIGs:3464)	cirsim –r: 0.04s	cirsim –r: 0.02s
Sim13 (# of PIs + # of AIGs:85067)	12736 patterns simulated cirsim –r: 7.29s	19168 patterns simulated cirsim –r: 3.41s

Use cirsim –f pattern.xx to generate same amount of pattern and cirfraig to test the performance of fraig

File	My program	Reference program
Sim09 (# of PIs + # of AIGs:3464)	1920 patterns simulated cirfraig: 0.21s	1920 patterns simulated cirfraig: 0.2s
Sim13 (# of PIs + # of AIGs:85067)	22912 patterns simulated cirfraig: 259.4s	22912 patterns simulated cirfraig: 120.5s

Analysis:

The performance of simulation can be further improved by event-driven method (a pre-procedure that discard the same input pattern)

The performance of fraig is really bad, and it’s predictable because I just use brutal method to call minisat engine and fraig the circuit.
It can be accelerated by :

- Split the FEC groups during the fraig operation.
- Find the special input pattern that SAT (use function getValue()), and use it to simplify the fraig operation.
- Prove the FEC groups from the deepest PI and gates to PO. (DFS sequence)

2. Correctness

The FEC groups collected after simulation (using the file-specified pattern) is exactly same as the FEC groups collected by the reference program.

■ Epilogue

DSnP class shows me the not fancy but practical usage of C++ programs.
I tried hard to make my program clean and improve its scalability.
The final project is a little bit above my limit. But I learned a lot.
Thanks to my teacher Ric and all the friends that work with me together.