Core refinement heuristic

Xiaojun Sun, xiaojuns@ece.utah.edu

February 27, 2016

1 The GB-core algorithm

Algorithm 1 GB-core algorithm (based on Buchberger's algorithm)

Input: $F = \{f_1, \ldots, f_s\} \in \mathbb{F}[x_1, \ldots, x_n], f_i \neq 0$
Output: List of coefficient-polynomial pairs $Q = \{(c_1, f_1), \ldots, (c_m, f_m)\}, f_i \in F$ where $1 \leq i \leq m$

Initialize: list $G \leftarrow F$; queue $P \leftarrow \emptyset$; hash table $H \leftarrow \emptyset$
for $i \leftarrow 2 \ldots s$ **do**
  for $j \leftarrow 1 \ldots i-1$ **do**
    P.push(pair (f_i, f_j))
  end for
end for
while $P \neq \emptyset$ **do**
  list of poly-coefficient pairs $Q \leftarrow \emptyset$
  $(f, g) \leftarrow P$.pop()
  $sp, (c1, f), (c2, g) \leftarrow$ spoly-pair(f, g)
  Q.push$((c1, f), (c2, g))$
  $r, Q \leftarrow$Reduce(sp, G, Q)
  if $r \neq 0$ **then**
    $P \leftarrow P \cup \{(g, r) \mid \forall g \in G\}$
    $H \leftarrow H \cup$ key-value pair (r, Q)
    $G \leftarrow G \cup r$
  end if
  if $r = 1$ **then**
    $Q \leftarrow H(key = 1)$
    while $(c, f) \in Q$ and $f \notin F$ **do**
      Replace (c, f) with $H(key = f)$
    end while
    Return(Q)
  end if
end while

Reduce(polynomial f, set of polys $G = \{g_1, \ldots, g_m\}$, initial quotients list Q)
Initialize $r \leftarrow f$
while $G \neq \emptyset$ **do**
  poly $f \leftarrow r$
  for $i \leftarrow 1 \ldots m$ **do**
    if $LM(g_i) \mid LM(r)$ **then**
      Q.push$((g_i, \frac{LT(r)}{LT(g_i)}))$
      $r \leftarrow r - \frac{LT(r)}{LT(g_i)} \cdot g_i$
      Break;
    end if
  end for
  if $r = f$ **then**
    Return (r, Q)
  end if
end while

2 UNSAT core refinement

As all other MUS extractors, our algorithm cannot generate a minimal (irreducible) core in one shot. To get a smaller core, we need to restart our tool with the current core we have. In this section several heuristics are applied to our algorithm to increase the probability to generate a smaller core in next iteration.

2.1 Leaf polynomial cancellation due to quotients

First of all, before we start a new iteration of our GB engine, we can analyze the information we get from last iteration and remove redundancy. In the last section we extract an UNSAT core by including all leaf polynomials in our tree to refutation "1". The tree represents a linear combination with polynomial coefficients of leaf polynomials. For example, in Fig.**??** refutation "1" can be written as equation

$$1 = c \cdot f_2 + f_5 + f_3 + f_4 + f_1 + ac \cdot f_2 + a \cdot f_2 + f_3 + c \cdot f_2 + f_2$$

By combining like terms, the RHS polynomial can be simplified as

$$1 = f_1 + (ac + a + 1)f_2 + f_4 + f_5$$

Note that polynomial f_3 is eliminated as $f_3 + f_3 = 0 \pmod 2$. Thus f_3 is redundant in the core we previously extracted.

2.2 Refining heuristics

After eliminating all redundant polynomials, we can call our GB engine with the new core. A effective heuristic should enhance the probability that the refutation "1" is composed by fewer polynomials. In our GB computing algorithm, we use a strategy to pick critical pairs such that polynomials with larger indices get involved as late as possible:

$$(f_1, f_2) \to (f_1, f_3) \to (f_2, f_3) \to (f_1, f_4) \to (f_2, f_4) \to \cdots$$

Meanwhile, for the reduction process $Spoly(f_i, f_j) \xrightarrow{F}_+ f_{new}$, we pick divisor polynomials from F following the order of polynomial indices. Therefore, by renaming the polynomial indices and moving specific polynomials to small indices, we can increase the likelihood they are selected into the UNSAT core.

A straightforward inspiration is, by moving polynomials that are most likely included in the refutation tree to small indices, we can reach a fixpoint with fewer calls to GB engine. In this paper, we use 2 criteria to increase the probability that a polynomial is included in the refutation tree. The one is the "distance" to refutation "1", the other is the "frequency" a polynomial appears in refutation tree.

"Distance" in a refutation tree corresponds to the number of arcs on the shortest path from refutation "1" to leaf polynomial Usually polynomials with

shorter distance to "1" are used as divisors in later steps of polynomial reductions, which indicates they will more generally have lower degree leading terms. If we use a degree-lexicographic order, then term cancellations reduce the degree of the polynomial. So polynomials with lower degree leading terms are more likely to be used for reduction, such that the possibility they appear in refutation tree is larger. Similarly, the motivation for "frequency" of appearance is as follows: polynomials appearing frequently in refutation tree also indicates they have some properties make them "favourable" in UNSAT core selection. For example, their leading terms may contain a variable that suppose to be included in a necessary clause of a minimal core.

Our tool applied both heuristics: we sort the polynomials in the core by "distance" criterion, then use frequency as tiebreaker – if there are multiple polynomials with same "distance", we put the one appears most frequently in last refutation tree first. We use an example to illustrate our heuristic.

Example: Consider a set of 6 polynomials which is UNSAT:

$$
\begin{aligned}
&f_1 : x_1x_3 + x_3\\
&f_2 : x_2 + 1\\
&f_3 : x_2x_3 + x_2\\
&f_4 : x_2x_3\\
&f_5 : x_2x_3 + x_2 + x_3 + 1\\
&f_6 : x_1x_2x_3 + x_1x_3
\end{aligned}
$$

After the first iteration, we get a tree of refutation in Fig.1(a). "Distance"

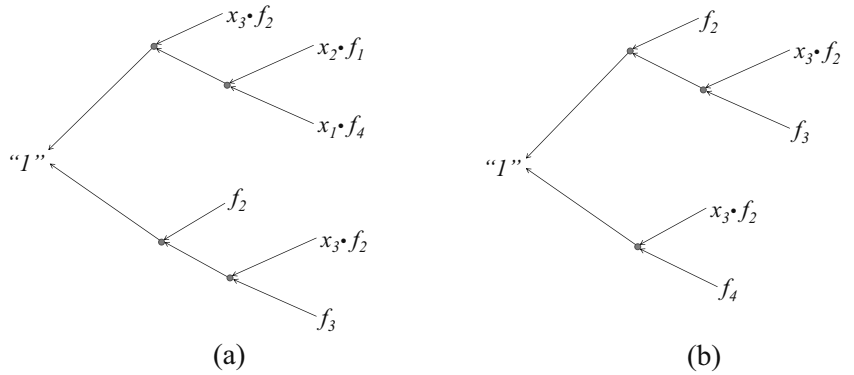

Figure 1: Refutation trees of core refinement example

from f_2 to "1" is 2, so we put it in first place. The "distance" and "frequency" for other polynomials are identical, so we keep their ordering untouched. We

re-index the polynomial set

$$f'_1 = f_2, f'_2 = f_1, f'_3 = f_3, f'_4 = f_4$$

and apply our GB-core algorithm on $\{f'_1, f'_2, f'_3, f'_4\}$. The result is as Fig.1(b) shows. We reach a fixpoint, and the core $\{f'_1, f'_3, f'_4\}$ i.e. $\{f_2, f_3, f_4\}$ is proved to be minimal.

3 Experiment results

We have implemented our approach using the SINGULAR symbolic algebra computation system [v. 3-1-6] [1]. With our tool implementation, we have performed experiments to extract a minimal UNSAT core from a given set of polynomials. Our experiments run on a desktop with 3.5GHz Intel Core^{TM} i7-4770K Quad-core CPU, 16 GB RAM and 64-bit Ubuntu Linux OS.

Nowadays most SAT benchmarks are huge, which choke our GB engine. In our experiment we use our customized benchmark library: "aim-100" is revised from ramdon 3-SAT benchmark "aim-50/100", "subset" is generated for some random subset sum problem, "cocktail" is revised from a mixed factorization/random 3-SAT benchmark, and "phole4/5" are generated by added some redundancies to pigeon hole benchmarks. Meanwhile we create "$3 \times 3/5 \times 5$ SMPO" benchmarks by translating miter circuits of unrolled sequential normal basis multiplier [2] and its specification circuit, and "2×2 MasVMon" benchmark is the miter circuit of Mastrovito multiplier and Montgomery multiplier. These benchmarks are in moderate size for our GB engine but not too trivial.

Table 1: Results of running benchmarks using our tool

Benchmark	# Polys	# MUS	Size of core by our tool	# calls of GB engine	# redundant polynomials by quotient cancellation	Runtime (sec)
5×5 SMPO	240	137	137	8	57	3160
aim-100	79	22	22	1	0	43
3×3 SMPO	17	2	2	1	0	0.07
2×2 MasVMont	27	23	23	2	0	2.3
subset	100	6	6	1	0	2.4
cocktail	135	4	4	2	1	5
phole4	104	10	16	1	0	4.4
phole5	169	19	25	3	2	14

In table 1, we list the details of our experiment results. It shows in most cases, our tool can get to the minimal core when hitting a fixpoint. Meanwhile, with a limited number of calls to our GB engine, our tool can remove a major part of redundancies which indicates our tool can extract a relatively small core efficiently.

From the experiment results we can also make several observations. One observation is our tool generates minimal core when reaching fixpoint, which reflects the effectiveness of our heuristic. Another fact is, for some test instances, quite a few redundant polynomials are removed before we start calling the GB engine. This shows the strong potential for eliminating generator polynomials by our "cancellation due to quotients" heuristic.

References

[1] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann, "SINGULAR 3-1-6 — A computer algebra system for polynomial computations," http://www.singular.uni-kl.de, 2012.

[2] A. Reyhani-Masoleh and M. A. Hasan, "Low complexity word-level sequential normal basis multipliers," *Computers, IEEE Transactions on*, vol. 54, no. 2, pp. 98–110, 2005.