MUS-driven Synthesis of Propagation Complete Encodings

Arunothia Marappan[1] Prof. Harald Sondergaard[2] Dr. Graeme Gange[2]
(1) Indian Institute of Technology Kanpur (2) University of Melbourne

Abstract

In this work, we present an algorithm to compute Propagation Complete Encoding (PCE) of any given input encoding. Our algorithm is inspired by the idea of minimal unsatisfiable cores/sets (MUSes) and is practically faster, more intuitive and simpler to implement than the previously presented ones.

Unit Propagation

- $(x \lor y \lor z) \land (\neg x \lor z) \land (x)$
- $(x \lor y \lor z) \land (\neg x \lor z) \land (x)$
- $(x \lor y \lor z) \land (\neg x \lor z) \land (x)$
- $(x \lor y \lor z) \land (\neg x \lor z) \land (x)$

Propagation Completeness

- x = if b then y else z
- $(b \longrightarrow (x \longleftrightarrow y)) \land (\neg b \longrightarrow (x \longleftrightarrow z))$
- $(\neg b \lor \neg x \lor y) \land (\neg b \lor \neg y \lor x) \land (b \lor \neg x \lor z) \land (b \lor \neg z \lor x)$
- $(\neg b \lor \neg x \lor y) \land (\neg b \lor \neg y \lor x) \land (b \lor \neg x \lor z) \land (b \lor \neg z \lor x) \land (\neg x \lor y \lor z) \land (x \lor \neg y \lor \neg z)$
- $((y \land z) \longrightarrow x) \land ((\neg y \land \neg z) \longrightarrow \neg x)$

Minimal Unsatisfiable Sets (MUSes)

- $(x \lor \neg y) \land (\neg x \lor y \lor z) \land (x \lor y \lor \neg z)$
- x — False
  y — True
  z — False
- x — False
  y — True
  z—?

References

[1] Martin Brain, Liana Hadarean, Daniel Kroening, and Ruben Martins.
Automatic generation of propagation complete sat encodings.
In *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, volume 9583 of *Lecture Notes in Computer Science*, pages 536–556. Springer, 2016.

[2] Mark H Liffiton and Ammar Malik.
Enumerating infeasibility: Finding multiple muses quickly.
In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 160–175. Springer Berlin Heidelberg, 2013.

Existing Algorithm

Algorithm 3: Brain's Algorithm (Brain et al. 2016)

Input : $< \Sigma, E_0, E_{Ref} >$
Output: E

E $\longleftarrow E_0$
PQ.push($\lambda v.?$)
while *not PQ.empty()* **do**
  $pa \longleftarrow$ PQ.pop()
  foreach $v \in x | x \in \Sigma$ *and* $UP(E)(pa)(v) = ?$ **do**
    foreach $l \in v, \neg v$ **do**
      $pa' \longleftarrow pa \sqcap assign(l)$
      if $SATSolver(E_{Ref}, pa') = SAT$ **then**
        PQ.push(pa')
      else
        $E \longleftarrow E \cup \{\neg MUS(pa', E_{Ref})\}$
        PQ.compact()
    end
  end
end

Proposed Algorithm

Algorithm 4: MUS-driven Synthesis of PCE

Input : <intLst, eRef>
Output: redundancyRemover(pce)

$\varphi = \bigwedge_{x \in intLst} (\neg x^T \lor \neg x^F)$
while *φ is satisfiable* **do**
  mu = solution(φ)
  if *isSAT(eRef $\land$ mu)* **then**
    muSAT = solution(eRef $\land$ mu)
    $\varphi = \varphi \land$ (negSAT (MUS(muSAT, $\neg$ eRef)))
  else
    $\varphi = \varphi \land$ (negUnSAT (MUS(mu, eRef)))
    pce = pce $\land$ ($\neg MUS$(mu, eRef))
end

Conclusion

- The proposed algorithm prunes in both the cases, making it more efficient.
- Any tautology of size **n**, takes $\mathbf{3^n}$ steps in Brain's Algorithm ([1]) whereas can be solved in constant steps in our algorithm.

Results

Gadget Name	Number of Clauses [Reported - [1]]	Number of Clauses [Algorithm-3] (Haskell)	Time (seconds) [Algorithm-3] (Haskell)	Number of Clauses [Algorithm-4] (Haskell)	Time (seconds) [Algorithm-4] (Haskell)
ult-gadget	6	6	0	6	0
slt-gadget	6	6	0	6	0
full-add	14	14	0	12	0
bc3to2	76	72	1	57	0
bc7to3	254	254	48	196	17
mult2	19	17	0	12	0
mult-const3	20	12	0	6	0
mult-const5	24	20	0	11	0
mult-const7	32	26	0	19	0
ult-6bit	158	158	2145	158	124
add-3bit	96	96	4	84	0
add-4bit	336	336	264	288	5
bc3to2-3bit	1536	1536	2044	1536	65
mult-4bit	670	640	451	594	38

Submitted to Grace Hopper Celebration India - 2016 (GHCI'16) Poster Session held on 08/12/2016 in Bangalore