Generating Propagation Complete Encodings in Haskell

Arunothia Marappan

Under Guidance of Graeme Gange

Abstract

In this project, we implemented the algorithm given in [1] for finding the propagation complete encoding for any given input encoding. Propagation complete encodings are those encodings who when fed to a SAT Solver can efficiently prune away all the unyielding paths that could otherwise occur while performing unit propagation. The implementation has been done in Haskell.

THE UNIVERSITY OF

MELBOURNE

Computing and Information Systems
University of Melbourne

1 Implementation Details

1.1 Representing CNF

In our implementation, we represent a CNF encoding as a list of list of integers.

- The elements within the outer list are to be combined with an AND.

- The elements within the innner list are to be combined with an OR.

- Positive Integer i represents the literal X_i and negative integer i represents the literal $\neg X_i$.

- Integer 0 is not included in the representation and hence should be avoided.

1.2 Input Format

We assume the input to be a CNF encoding. The input arguments are

- An integer 'n', implying the vocabulary or variable set is X_1 to X_n

- A list of list of integers that will represent the CNF of E that is being computed (It will be E_0 at the start)

- A list of list of integers that will represent the CNF of E_{ref} (which is the CNF for which equi-satisfiable formula is to be found)

- A minHeap (priority queue) that is required by the algorithm for looping. It is not a user input argument. It should by default be a singleton queue with the fully undefined partial assignment.

1.3 Output Format

The output is

- List of list of Integers representing the encoding that is equi-satisfiable to E_{ref} and is propagation complete.

1.4 Methodology in Haskell

PAValue is the defined Data Type that can either be PATrue, PAFalse or PAQuest representing the values True, False or Question that a partial assignment can assign to a variable. We define Partial Assignments as Maybe[PAValue], where Nothing represents the contradicting partial assignment or the bottom. The ordering amongst Partial Assignments are established as - Nothing at the least and more undefined the partial assignment is, higher it is in the ordering. For the implementation of Priority Queues, we use Haskell Package Data.Heap. From the ordering of Partial Assignments mentioned, it is clear that we used maxHeap to mimic the priority queue desired. Sat Solver for the implementation has been taken from [2].

2 Examples

2.1 If-then-Else Gadget

$$x = if\ b\ then\ y\ else\ z$$

Which can be encoded as -

$$(b \longrightarrow (x \longleftrightarrow y)) \wedge (\neg b \longrightarrow (x \longleftrightarrow z))$$

Which in the CNF form corresponds to

$$(\neg b \vee \neg x \vee y) \wedge (\neg b \vee \neg y \vee x) \wedge (b \vee \neg x \vee z) \wedge (b \vee \neg z \vee x)$$

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] gatlin. sat.hs. https://gist.github.com/gatlin/1755736, 2012. [Online; accessed June-2016].