Decision procedures and verification - Backbones

Jaroslav Šafář

September 23, 2020

1 Theory and description of the algorithm

The task is to write a program which uses a SAT solver as an oracle and finds all backbones of a given CNF φ.

We say that a literal l is a backbone of φ if l is in every model of φ, or in other words, the value of l is implied by φ, i.e. $\varphi \models l$. We will use the following theorem:

$$\varphi \models l \Leftrightarrow \varphi \cup \{\neg l\} \text{ is unsatisfiable formula.} \tag{1}$$

We can also use the following observation:

1. If $\varphi \cup \{\neg l\}$ is UNSAT and $\varphi \cup \{l\}$ is SAT then l is a backbone of φ.
2. If $\varphi \cup \{l\}$ is UNSAT and $\varphi \cup \{\neg l\}$ is SAT then $\neg l$ is a backbone of φ.
3. If $\varphi \cup \{l\}$ is SAT and $\varphi \cup \{\neg l\}$ is SAT then neither l nor $\neg l$ is a backbone of φ.

One way to approach this problem is to actually find some model of φ (if it exists) and then go through the literals of this model. For every such literal l check whether $\varphi \cup \{\neg l\}$ is SAT or UNSAT. In the case of UNSAT we know that l is a backbone of φ and we can update the formula φ by $\varphi \cup \{l\}$. Otherwise l is not a backbone of φ and we can use the returned model of $\varphi \cup \{\neg l\}$ to update the candidates on backbones by intersecting them with this model. This process is described in the following algorithm [Alg. 1].

The order in which we pop the cadidate literals from Γ may greatly effect the total number of CDCL solver calls. We can represent Γ as a priority queue and always pop a literal with the highest priority. The priority, which was chosen, is the number of occurrences of the literal in the clauses.

Note that the algorithm guarantees that the loop iterates at most $|var(\varphi)|$ times. Hence, the algorithm performs at most $|var(\varphi)| + 1$ CDCL calls in total.

2 Experimental results

Some smaller Uniform Random-3-SAT examples from SATLIB Benchmark Problems were used for experimental evaluation of CDCL algorithm. In all cases, the algorithm found the resulting backbone literals, and the resulting number of CDCL calls and total CPU time can be found in the following table [Tab. 1]. We do not present the backbones themselves because, in larger examples, the lists are quite large.

Algorithm 1: Iterative algorithm for finding backbones of a given formula

Input : CNF formula φ
Output: Backbones of φ or UNSAT

$sat, model \leftarrow CDCL(\varphi)$
if $\neg sat$ **then**
    return $UNSAT$
$\Gamma \leftarrow model$
$Backbones \leftarrow \emptyset$
while $\Gamma \neq \emptyset$ **do**
    $l \leftarrow \Gamma.pop()$
    $sat, model \leftarrow CDCL(\varphi \cup \{\neg l\})$
    if $\neg sat$ **then**
        $Backbones \leftarrow Backbones \cup \{l\}$
        $\varphi \leftarrow \varphi \cup \{l\}$
    else
        $\Gamma \leftarrow \Gamma \cap model$
return $Backbones$

	calls	time
uf20-01.cnf	11	0.035312
uf20-02.cnf	16	0.034596
uf20-03.cnf	21	0.055656
uf20-04.cnf	21	0.044903
uf50-01.cnf	49	0.334398
uf50-02.cnf	51	0.389571
uf50-03.cnf	37	0.282256
uf50-04.cnf	49	0.335773
uf75-01.cnf	38	0.514691
uf75-02.cnf	24	0.591996
uf75-03.cnf	75	0.818545
uf75-04.cnf	74	1.591875
uf100-01.cnf	47	7.063968
uf100-02.cnf	68	3.767319
uf100-03.cnf	59	6.001157
uf100-04.cnf	91	9.231800
uf125-01.cnf	73	18.255582
uf125-02.cnf	124	11.310766
uf125-03.cnf	34	23.345212
uf125-04.cnf	59	15.476930

Table 1: Number of CDCL calls and total CPU time of the backbones algorithm