Augmenting Clause Learning with Implied Literals (Poster Presentation)

Arie Matsliah[1], Ashish Sabharwal[2], and Horst Samulowitz[2]

[1] IBM Research, Haifa, Israel
ariem@il.ibm.com
[2] IBM Watson Research Center, Yorktown Heights, USA
{ashish.sabharwal,samulowitz}@us.ibm.com

There exist various approaches in SAT solvers that aim at extending inference based on unit propagation. For instance, *probing* [5] simply applies unit propagation of literals at the root node in order to detect failed literals [3] or to populate literal implication lists. The latter information can then, for instance, be used to shrink clauses by *hidden literal elimination* (e.g., if $a \mapsto b$ then $(a \vee b \vee c)$ can be reduced to $(b \vee c)$; cf. [4]).

Here we propose to strengthen clause learning by dynamically inferring literals that the newly learned clause entails. We say that a literal l is an *implied literal* for a clause C if all literals of C entail l. For instance, if $a \mapsto d$, $\neg b \mapsto d$, and $c \mapsto d$, then $(a \vee \neg b \vee c)$ entails d. While this insight has already been exploited in several methods (e.g., variations of *hyper binary resolution* and *hidden literal elimination*), we apply it to clause learning: when the SAT solver derives a new conflict clause c, we check if the literals in c imply a single or multiple literals which can then be propagated as new unit literals.

In order to employ this technique we first need to generate implication lists $L(l) = \mathit{UnitPropagation}(l)$ for each literal l. This is done at the root node of the search tree before the solving process starts and then periodically during search. During this computation, we also add not yet existing binary clauses corresponding to $\forall l \in L(p) : \neg l \mapsto \neg p$. As one might expect, we detect failed literals as well and add and propagate their negations as new unit literals; we do the same for all literals in the intersection of $L(p)$ and $L(\neg p)$ for all $p \in F$. Once the implied literal lists for each literal are computed, we iterate over the clauses in the original theory F and propagate all implied literals as new unit clauses.

Ideally we would like to augment these lists with new implications whenever new clauses have been learned by the solver. However, this operation can be computationally expensive and we must control how frequently it is performed. While learned clauses are usually 'forgotten' over time, we hold on to the implication lists and only extend them, when possible. Note that this can enable inference that might not be explicitly captured in the current clausal theory.

Whenever a new clause C is derived, we check whether $I = \bigcap_{l \in C} L(l)$ is non-empty. If so, we add all $p \in I$ as new unit clauses and backtrack the search to the root (i.e., restart). We reduce the computational effort of computing intersections by considering learned clauses of only a bounded length.

Table 1. Performance of `Glucose` and `Glucose+IL` on the application category instances of SAT Competition 2011 (selected benchmark families and all instances)

Benchmarks	`Glucose`			`Glucose+IL`		
	% Solved	PAR10	#Dec. (G1k)	% Solved	PAR10	#Dec. (G1k)
Goldberg (8)	87.5	9,045	617,058	100.0	774	377,047
Grieu-VMPC (6)	83.3	12,715	8,109,876	100.0	1,286	4,809,073
Jarivsalo-AAAI10 (13)	100.0	485	522,951	100.0	224	218,455
Rintanen-Sokoban (6)	33.3	43,871	2,196,775	66.8	22,241	624,748
Competition 2011 (300)	71.3	19,280	472,059	72.0	18,778	377,712

We implemented this approach in `Glucose 2.0` [1] and evaluated the performance on all 300 instances of the application track of SAT Competition 2011 [6]. All experiments were conducted on 2.3 GHz AMD Opteron 6134 machines with eight 4-core CPUs and 64 GB memory, running Scientific Linux release 6.1. We used a time limit of 6,500 sec (which roughly corresponds to the 5,000 sec timeout used in the competition), and limited activation of our method to instances with at most 2,000,000 clauses and 200,000 literals appearing in binary clauses. Implied literals are computed after 150 restarts and from then on in a geometrically increasing manner with a factor of 1.2. All runs used identical parameters (e.g., maximum learned clause length to check for implied literals).

Table 1 summarizes the results. For both versions of `Glucose` we show the percentage of instances solved, the PAR10 score (penalized average runtime where instances that time out are penalized with 10x the timeout), and the geometric mean shifted by $1,000$ of the number of decisions made on the instances solved by both approaches. The first four benchmark families highlight the potential of our approach. On all shown measures, the impact of implied literals is quite dramatic. For instance, on the `Sokoban` benchmark, adding inference based on implied literals doubles the number of solved instances. The final row shows that the new approach does not degrade performance of the baseline solver. In fact, it is able to solve more instances while making 20% fewer decisions.

We have extended this approach to also learn binary clauses when all but one literal in the learned clause imply a literal. Note that this corresponds to a generalized version of *hyper-binary resolution* [2].

References

1. G. Audemard and L. Simon. Predicting learnt clauses quality in modern SAT solvers. *IJCAI*, 399-404, 2009.
2. F. Bacchus. Enhancing Davis Putnam with Extended Binary Clause Reasoning. *AAAI*, 2002.
3. Jon Freeman. Improvements to Propositional Satisfiability Search Algorithms. *PhD. Thesis*, University of Pennsylvania, 1995.
4. Marijn Heule, Matti Järvisalo, Armin Biere. Efficient CNF simplification based on binary implication graphs. *SAT*, 201–215, 2011.
5. Ines Lynce, Joao Marques-Silva. Probing-Based Preprocessing Techniques for Propositional Satisfiability. *ICTAI*, 105, 2003.
6. SAT Competition 2011. *http://www.satcomptition.org.*