GLUCOSER: a solver that introduces variables during search*.

Gilles Audemard
Univ. Lille-Nord de France
CRIL/CNRS UMR8188
Lens, F-62307
audemard@cril.fr

George Katsirelos
Univ. Lille-Nord de France
CRIL/CNRS UMR8188
Lens, F-62307
gkatsi@gmail.com

Laurent Simon
Univ. Paris-Sud
LRI/CNRS UMR 8623 / INRIA Saclay
Orsay, F-91405
simon@lri.fr

Abstract

GLUCOSER is a CDCL solver with the ability to introduce fresh variables during search. We describe the heuristics that are used to introduce these variables as well as other modifications that are necessary to integrate this into the CDCL framework. The techniques here were described in [Audemard *et al.*, 2010], while the ideas used in the CDCL part were described in [Audemard and Simon, 2009]. The solver is available from `http://www.lri.fr/~simon/glucose`.

1 Introduction

The performance of SAT solvers can be improved by two general methods: either designing better heuristics or using a more powerful underlying proof system. The operation of CDCL solvers is naturally understood as a resolution procedure and moreover the CDCL algorithm and resolution p-simulate each other. Thus, a natural candidate for a more powerful proof system to serve as a basis for a CDCL-like solver is extended resolution. Extended resolution allows the introduction of a fresh variable at any point, which is equivalent to a Boolean expression over the existing variables of the formula, including previously introduced variables. Unlike resolution, no superpolynomial lower bounds are known for extended resolution. On the other hand, no heuristics have been proposed on choosing appropriate variables to introduce.

In GLUCOSER, we have introduced a simple heuristic that aims to exploit the power of extended resolution in some cases, without compromising the efficiency of the CDCL solver.

As its name implies, GLUCOSER is based on GLUCOSE, which is in turn based on MINISAT. The heuristics and data structures that do not pertain to introducing fresh variables are identical to that solver.

2 Introducing new variables

The main insight we exploit in GLUCOSER is that, when a solver learns two clauses of the form $-l_1 \vee \alpha$ and $-l_2 \vee \alpha$, it may repeat a sequence of resolution steps that is identical for both clauses, if these steps do not involve the variables l_1 or l_2. We can avoid this repetition by introducing the fresh variable $x \iff l_1 \vee l_2$. Despite its simplicity, this scheme has several key advantages: first, it can be applied to clauses that are not part of the input formula, but are derived during search; second, repeated application of this rule can introduce variables that are equivalent to arbitrarily complex expressions. The former means that the scheme cannot be fooled by simple "disguising" schemes. In addition, it allows us to expore the derivation of the proof as a source of information for which variables to introduce. The latter means that this scheme can in some cases use the full power of extended resolution.

For the sake of simplicity, the above rule is only applied to successive pairs of clauses which differ only in the asserting literal. This decision was made both for efficiency but also based on the intuition that the heuristics of CDCL solvers are tuned to keep the solver exploring the same search space despite restarts. So, it is likely that pairs of clauses that match the scheme $-l_1 \vee \alpha$, $-l_2 \vee \alpha$ will be discovered close to each other. Therefore, it would be sufficient to examine only a small window of recent clauses to detect many such pairs of clauses. By pushing this reasoning to its extreme and setting the window size to 1, we get the heuristic described above.

If n successive learnt clauses have the form $-l_i \vee C$ ($1 \leq i \leq n$) then we add the rules $z_1 \iff l_1 \vee l_2$, but also all $z_i \iff l_i \vee l_{i+1}$ for $2 \leq i < n$. Another alternative may be to add $z_i \iff z_i \vee l_{i+1}$ instead of $z_i \iff l_i \vee l_{i+1}$ but we found empirically that this performed worse.

Finally, when new clauses are introduced to encode the expression $x \iff l_1 \vee l_2$, the solves needs to ensure that it modifies the assignment so that no unit propagations are missed. However, because of our restrictions, this case will never happen. Indeed, consider again two successive learnt clauses $-l_1 \vee \alpha$ and $-l_2 \vee \alpha$. We add the three clauses that encode $z \iff l_1 \vee l_2$. Then, because l_1 and l_2 were UIP literals, $-z \vee l_1 \vee l_2$ will be true and z will be propagated to true according to $z \vee -l_1$. Hence, our restriction over detected pairs of clauses also ensures that the desirable property of learnt clauses that they assert a literal on backtracking still holds even when introducing new variables and additional clauses at a given conflict. In particular, we don't force the CDCL solver to restart earlier, or to particularly reorder its

*supported by ANR UNLOC project n° ANR-08-BLAN-0289-01

decision dependencies.

3 Extended variables in new clauses

As soon as a fresh variable $z \iff l_1 \vee l_2$ is introduced, we have to ensure that we replace new clauses in the remaining proof that match the form $l_1 \vee l_2 \vee \beta$ with $z \vee \beta$. This systematic reduction is important because this allows the new variable z to be propagated even when the learnt clause $l_1 \vee l_2 \vee \beta$ would not have been. For example, if all literals in β are false, the clause $l_1 \vee l_2 \vee \beta$ is not unit, but $z \vee \beta$ is. Empirically, we discovered that if we do not perform this reduction step, then z will almost never occur in conflict analysis, so z will not be used in the resolution proof that is produced. Note that we restrict the application of this reduction step to clauses learnt after we introduce z, mostly for efficiency.

To support this reduction step, we maintain a hash table of pairs of extended literals, and we probe the hash table each time a conflict is performed to replace pairs of literals by their extended variable. It has to be noticed that the use of such a hash table implies a special case were the reduction introduces a choice. Suppose we learn the clause $C \equiv l_1 \vee l_2 \vee l_3 \vee \beta$ and we have previously introduced the two new variables z_1 and z_2 such that $z_1 \iff l_1 \vee l_2$ and $z_2 \iff l_2 \vee l_3$. Then our iterative procedure of reduction will have to make a choice between the two clauses $C_1 \equiv z_1 \vee l_3 \vee \beta$ or $C_2 \equiv l_1 \vee z_2 \vee \beta$. This choice is handled by prefering to use extended variables with a higher VSIDS score at the time of the reduction.

4 Variable deletion

In analogy to the way that the learnt clause database is regularly reduced during search, we delete unused variables to keep the cost of unit propagation from dominating the runtime. In particular, we maintain a dependency graph of introduced variables. Whenever the clause database is reduced, we also remove those leaves of the dependency graph whose VSIDS score is less than the median among the introduced variables. A low VSIDS score indicates that the introduced variables are not used during resolution, thus do not contribute to producing a shorter proof. Additionally, any learned clauses that contain these variables are also deleted with a linear scan of the database, which happens anyway during clause deletion.

5 Conclusion

GLUCOSER is a solver that attempts to use some of the power of extended resolution in the CDCL framework. The aim in our design was that introducing fresh variables does not hinder the efficiency of a CDCL solver. The resulting solver exhibits better scaling in some instances that are known to be hard for resolution, such as the `urq` family. The framework for introducing variables is simple and relatively self contained so can be expected to be easy to use in other solvers. We expect to gain further improvements by exploiting other local forms of redundancy in the resolution proofs generated by CDCL solvers.

References

[Audemard and Simon, 2009] G. Audemard and L. Simon. Predicting learnt clauses quality in modern sat solvers. In *proceedings of IJCAI*, 2009.

[Audemard *et al.*, 2010] G. Audemard, G. Katsirelos, and L. Simon. A restriction of extended resolution for clause learning sat solvers. In *proceedings of AAAI*, 2010.