Glucose_nbSat and Glucose_nbSatRsltn

Chu Min LI*†, Fan Xiao* and Ruchu XU*
*Huazhong University of Science and Technology, China
†MIS,Universit de Picardie Jules Verne, France

I. Introduction

In this paper, we present two derived versions of Glucose called *Glucose_nbSat* and *Glucose_nbSatRsltn*. Glucose_nbSat and Glucose_nbSatRsltn are developed from the code source of Glucose-3.0[1], [2] by implementing a significant change in the learnt clause database management and a limited redundant clause simplification and removing. Glucose_nbSatRsltn and Glucose_nbSat are only a little different. We first describe the main features of Glucose relevant to our changes, and then describe Glucose_nbSat and Glucose_nbSatRsltn.

II. Main features of Glucose

Glucose is a very efficient CDCL-based complete SAT solver. It is always one of the awarded winning SAT solvers in SAT competitions (challenge, race) since 2009. The main features of Glucose and its updated versions include a measurement of learnt clause usefulness called LBD and used in the cleaning of the learnt clause database. In the recent versions of Glucose such as Glucose-3.0, once the number of clauses learnt since the last database cleaning reaches 2000 + 300*x, where x is the number of database cleanings performed so far, the cleaning process is fired (i.e., the learnt clauses are sorted in the decreasing order of their LBD, and the first half of learnt clauses are removed except binary clauses, clauses whose LBD value is equal to 2, and the clauses that are reasons of the current partial assignment. In addition, Glucose-3.0 uses a very aggressive restart strategy [3], in such a way that the solver is very frequently restarted.

Our new learnt clause database management is based on the above features of Glucose-3.0.

III. Cleaning learnt clause database at the root of a search tree

A CDCL based SAT solver usually uses the restart mechanism [4], every restart constructing a binary search tree from scratch. In Glucose, as well as in most CDCL-based solvers, a learnt clause database cleaning process can be fired inside a binary search tree. Two observations can be made about this strategy: (1) there are locked clauses, i.e. clauses that are reasons of the current partial assignment, that cannot be removed, (2) the part of the tree before the cleaning and the part of the tree after the cleaning are constructed with very different learnt clause databases.

Glucose_nbSat differs from Glucose in that Glucose_nbSat cleans the learnt clause database always at the beginning of each restart, i.e., at the root of the search tree that is going to be constructed, when the number of learnt clauses becomes bigger or equal to 2000 + 300*x since the last database cleaning. In this way, clauses satisfied by variables fixed at the root are simply removed, as well as the literals falsified in the remaining clauses. Note that no clause is locked at the root of a search tree. Moreover, since the cleaning is not done inside the search tree, the search tree is constructed with the same incremental learnt clause database.

Compared with Glucose, the database cleaning is delayed in Glucose_nbSat, because it is not fired as soon as the number of the newly learnt clauses reaches a limit, but should wait for the next restart. However the delay is not important, since Glucose_nbSat performs fast restart as Glucose.

IV. Using a new measurement to predict the learnt clause usefulness

Modern CDCL-based SAT solvers usually save the last truth value of each variable when backtracking. When a free variable is picked as a decision variable, it is assigned the saved value. It is easy to see that at least one clause in which literals are all falsified by the saved assignment will become unit and change the saved value of a variable during unit propagation. More generally, a clause has more chance to become unit if the number of literals satisfied by the current saved truth value is smaller. On the contrary, those clauses with many literals satisfied by the saved truth value have little chance to become unit and should be removed.

Based on the above observation, we introduce a new measurement to predict the usefulness of a learnt clause, namely the number of literals satisfied by the saved assignment, denoted by *nbSat*, and implement the following learnt clause database cleaning strategy in Glucose_nbSat:

1) compute the number of literals satisfied by the saved assignment for each learnt clause, denoted by *nbSat*;
2) Sort all learnt clauses in the decreasing order of their nbSat value, breaking ties using the decreasing order of their LBD value. The remaining ties are broken using the clause activity value as in Glucose.
3) Remove the first half of learnt clauses (i.e. those with bigger nbSat values), by keeping binary clauses and clauses whose LBD is 2 as in Glucose.

Note that the saved assignment changes frequently during search. The measurement nbSat works only when the learnt clause database cleaning is fired frequently, because otherwise, it does not reflect the current search state after many conflicts. This is not a problem with Glucose_nbSat, because Glucose_nbSat cleans the database frequently as Glucose, making it relevant to use the nbSat measurement in the database cleaning.

V. Other embedded techniques

When a learnt clause is in the first half after all learnt clauses are sorted in the decreasing order of their nbSat value, i.e., when it is going to be removed by the database cleaning process, we check if it subsumes an original clause or if it can be resolved with an original clause to produce a resolvent that subsumes the original clause. In the first case, the learnt clause replaces the original clause and will never be removed. In the second case, the produced resolvent replaces the original clause and will never be removed.

Example. Let $x_1 \vee x_2 \vee x_3 \vee x_4$ be an original clause, and $\bar{x}_2 \vee x_3 \vee x_4$ be a learnt clause, then the resolvent $x_1 \vee x_3 \vee x_4$ is added as an original clause that is never removed, and $x_1 \vee x_2 \vee x_3 \vee x_4$ is removed.

The above process is also applied to simplify the set of original clauses as a preprocessing in Glucose_nbSat. More concretely, the original clauses are sorted in the decreasing order of their size: c_1, c_2, ..., c_m. Each c_i $(1 \leq i \leq m)$ is checked if there is a c_k $(k<i)$ such that c_i subsumes c_k or if c_i and c_k can be resolved to produce a resolvent that subsumes c_k. In both cases, c_k is removed. The resolvent in the second case is inserted in the set of original clauses.

There is only a different between Glucose_nbSat and Glucose_nbSatRsltn. In Glucose_nbSat, if a learnt clause can replaces the original clause and is inserted in the database, we remove one more learnt clause in reduceDB().

References

[1] G. Audemard and L. Simon, "Glucose: a solver that predicts learnt clauses quality," *IJCAI'09*, 2009.

[2] ——, "Glucose in the sat 2014 competition," *in Proceedings of the 2014 SAT competition*, 2014.

[3] ——, "Refining restarts strategies for sat and unsat formulae," *in Proceedings of the 22nd International Conference on Principles and Practice of Constraint Programming (CP-12)*, 2012.

[4] C. P. Gomes, B. Selman, and K. Henry, "Boosting combinatorial search through randomization," *in Proc. AAAI-98*, Madison, WI, July 1998.