# GLUEMINISAT2.2.8

Hidetomo Nabeshima
University of Yamanashi, JAPAN

Koji Iwanuma
University of Yamanashi, JAPAN

Katsumi Inoue
National Institute of Informatics, JAPAN

***Abstract*—GLUEMINISAT is a SAT solver based on MINISAT 2.2 and the LBD evaluation criteria of learned clauses. The main feature of the version 2.2.8 is the on-the-fly lazy simplification techniques which consists of various probing techniques, self-subsuming resolution, on-demand addition of binary resolvents, and clause minimization by binary resolvents. The base solver has features used in GLUCOSE 3.0 such as restart blocking, fluctuation strategy in the initial stage of search, additional maintaining of good learned clauses, etc.**

## I. INTRODUCTION

GLUEMINISAT is a SAT solver based on MINISAT 2.2 [1] and the LBD evaluation criteria of learned clauses [2]. We have re-implemented the version 2.2.8 from MINISAT2.2 to evaluate the performance of the on-the-fly lazy simplification techniques [3], which consists of the following techniques:

1) variable elimination and equivalent literal substitution based on probing techniques [4],
2) original clause and learned clause simplification by self-subsuming resolution with binary resolvents, and
3) on-demand addition of binary resolvents.

The computational cost of these techniques is negligibly small. Hence, these techniques are executed frequently throughout the process of modern conflict-driven clause learning (CDCL) solvers, that is, unit propagation, conflict analysis, removal of satisfied clauses, etc.

The base solver has features used in GLUCOSE 3.0 such as restart blocking [5], fluctuation strategy in the initial stage of search, additional maintaining of good learned clauses, etc.

## II. MAIN TECHNIQUES

GLUEMINISAT 2.2.8 has the on-the-fly lazy simplification techniques [3]. These techniques are defined as operations on binary resolvents, which are extracted from unit propagation process with almost no overhead [6], [7]. In GLUEMINISAT, the number of binary resolvents to be preserved is restricted and they are maintained in a simple data structure. For each literal $y$, we hold only one premise literal $x$ such that $\phi \models x \rightarrow y$, where $\phi$ is a given formula. We represent a premise literal of $y$ as *premise*$[y]$ (that is, $\phi \models$ *premise*$[y] \rightarrow y$). Initially, *premise*$[y] = y$. In the unit propagation process, if $y$ is propagated and it has a dominator [6], then the value of *premise*$[y]$ is updated with the dominator. In our experiments, each entry of premise is updated approximately 1000 times on average for solving 1192 application instances of SAT 2009 and 2011 competitions and SAT Challenge 2012 within 1200 CPU seconds. The preserved premise literals are updated frequently during search. That is, we maintain a partial snapshot of binary resolvents which evolves during search. This variation of premise literals contributes to the realization of low cost simplification techniques which are executed on-the-fly.

We can execute probing techniques [4] with a constant time by using the premise literals. For example, the necessary assignment probing can be represented as follows: suppose that $\phi$ is a formula and $x, y$ are literals. If $\phi \models x \rightarrow y$ and $\phi \models \neg x \rightarrow y$, then $\phi \models y$. This probing technique requires two premise literals of $y$. We can get two premise literals of $y$, that is, the old value of *premise*$[y]$ before updating of it and the new value of it. We denote the old and new values as *oldpremise*$_y$ and *newpremise*$_y$, respectively. Then, we can execute the necessary assignment probing as follows: if *oldpremise*$_y = \neg$*newpremise*$_y$, then $\phi \models y$ holds. Other probing techniques can be executed in the same way [3]. GLUEMINISAT executes these on-the-fly probing techniques when an entry of the array *premise* is changed.

We can simplify clauses based on binary self-subsumption by using the array *premise*. Given a clause $C$ and two literals $x, y \in C$, we define that $x$ *is redundant by* $y$ *in* $C$ if *premise*$[y] = x$ or *premise*$[\neg x] = \neg y$, since the resolvent of $C$ and $x \rightarrow y$ is $C \setminus \{x\}$ and it subsumes $C$. The redundant literals can be eliminated from a clause. Let $C$ be a clause $\{w_1, w_2, x_1, \cdots, x_n\}$, where $w_1$ and $w_2$ mean watched literals and $x_i$ is an unwatched literal in two watched literal schema [8]. In GLUEMINISAT, we check whether $x_i$ is redundant by $w_1$ or $w_2$ to avoid the updating cost of the list of watched clauses (if $w_i$ is eliminated, then we should update the list of watched clauses). This binary self-subsumption checking can be incorporated with the scanning-loop of literals in a clause, such as conflict analysis and removal of satisfied clauses.

Suppose that $\phi$ is a formula and $\alpha$ is an assignment. UP$(\phi, \alpha)$ represents the assignment after the unit propagation process. Let be *premise*$[y] = x$. This means that $y \in$ UP$(\phi, \{x\})$, but the contrapositive $\neg x \in$ UP$(\phi, \{\neg y\})$ may not hold. This is because the binary resolvent $x \rightarrow y$ does not exist as a clause in $\phi$ explicitly. To enhance the unit propagation, for each literal $p \in$ UP$(\phi, \alpha)$, if *premise*$[\neg p] \notin$ UP$(\phi, \alpha)$, then the binary resolvent *premise*$[\neg p] \rightarrow \neg p$ is added to $\phi$ as a clause. This on-demand addition is executed after the unit propagation process.

Suppose that $C = \{x_1, \cdots, x_n\}$. If there is a implication chain such that $\neg x_i \rightarrow \cdots \rightarrow \neg x_j$ $(i \neq j)$, then $x_j$ can be eliminated from $C$. This is a generalization of the above binary self-subsumption checking and a special case of minimization technique used in MINISAT [9]. GLUEMINISAT

executes this checking for each learned clause after conflict analysis. For each literal $x_j \in C$, GLUEMINISAT tries to construct a chain $\cdots \rightarrow y_2 \rightarrow y_1 \rightarrow \neg x_j$ from the end, where $y_1 = \mathit{premise}[\neg x_j]$ and $y_{k+1} = \mathit{premise}[y_k]$ $(k \geq 1)$. This construction is continued until $y_l$ is $\neg x_i$, where $x_j \in C$, (successful case) or the current assignment of $y_k$ is not true (failed case). The latter condition is introduced as a heuristics to avoid generating a long chain.

## III. OTHER TECHNIQUES

GLUEMINISAT has features which are implemented in GLUCOSE 3.0. The restart blocking [5] helps to catch a chance of making satisfying assignment. This strategy postpones restarting when the local trail size per a conflict is exceedingly greater than the global one.

The variable activity decay factor is one of parameters of VSIDS decision heuristics [10] used in MINISAT[1]. When the value of this parameter is small, the activity of recently unused variables decays quickly. This makes easy to move the search space, since the variable selection is not caught in the past activity. GLUCOSE 3.0 increases this parameter by 0.01 from 0.8 until 0.95 whenever 5000 conflicts occur. In our experiments, this strategy is effective for satisfiable instances. GLUEMINISAT also uses this strategy.

GLUCOSE 3.0 protects learned clauses from clause-deletion only once when the LBD value of those clauses decreases and are lower than a certain threshold. In the clause-deletion, basically it removes half of learned clauses in order of LBD values. But protected clauses survive only once. Furthermore, when good learned clauses whose LBD is less than or equal to 3 exist more than half, the limit of number of remained learned clauses is relaxed by adding a constant number. GLUEMINISAT follows this management strategy.

## IV. SAT COMPETITION 2014 SPECIFICS

GLUEMINISAT is submitted to Sequential, Application SAT+UNSAT track and Hard-combinatorial SAT+UNSAT track. The version 2.2.8 does not have a function to output an UNSAT proof. The previous version 2.2.7 which was submitted to SAT 2013 competition can output UNSAT proof when some simplification techniques are disabled.

## V. AVAILABILITY

GLUEMINISAT is developed based on MINISAT 2.2. Permissions and copyrights of GLUEMINISAT are exactly the same as MINISAT. GLUEMINISAT can be downloaded at http://glueminisat.nabelab.org/.

## ACKNOWLEDGMENT

This research is supported in part by Grant-in-Aid for Scientific Research (No. 24300007) from Japan Society for the Promotion of Science and by Kayamori Foundation of Informational Science Advancement.

## REFERENCES

[1] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003), LNCS 2919*, 2003, pp. 502–518.

[2] G. Audemard and L. Simon, "Predicting learnt clauses quality in modern SAT solvers," in *Proceedings of IJCAI-2009*, 2009, pp. 399–404.

[3] H. Nabeshima, K. Iwanuma, and K. Inoue, "On-the-fly lazy clause simplification based on binary resolvents," in *ICTAI*. IEEE, 2013, pp. 987–995.

[4] I. Lynce and J. P. Marques-Silva, "Probing-based preprocessing techniques for propositional satisfiability," in *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2003)*, 2003, pp. 105–110.

[5] G. Audemard and L. Simon, "Refining restarts strategies for sat and unsat," in *CP*, ser. Lecture Notes in Computer Science, M. Milano, Ed., vol. 7514. Springer, 2012, pp. 118–126.

[6] H. Han, H. Jin, and F. Somenzi, "Clause simplification through dominator analysis," in *Proceedings of Design, Automation and Test in Europe (DATE 2011)*, 2011, pp. 143–148.

[7] M. Heule, M. Järvisalo, and A. Biere, "Revisiting hyper binary resolution," in *Proceedings of the 10th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2013), LNCS 7874*, 2013, pp. 77–93.

[8] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the 38th Design Automation Conference (DAC 2001)*, 2001, pp. 530–535.

[9] N. Sörensson and A. Biere, "Minimizing learned clauses," in *Proceedings of SAT-2009*, 2009, pp. 237–243.

[10] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," in *Proceedings of DAC-01*, 2001, pp. 530–535.