The Yaga SMT Solver in SMT-COMP 2023

Martin Blicha[1,2], Drahomír Hanák[1], and Jan Kofroň[1]

[1] Charles University, Prague, Czech Republic
[2] Università della Svizzera italiana, Lugano, Switzerland

1 Overview

Yaga is a new SMT solver developed at Charles University with the goal to investigate alternatives to the dominant CDCL(T) framework for SMT solving. The solver implements the Model Constructing Satisfiability Calculus (MCSAT) [4, 7]. It currently has implementation of plugins for Boolean and rational variables which can be used to decide problems in quantifier-free linear real arithmetic. The Boolean plugin uses the typical mechanism of watched literals [8] to perform Boolean constraint propagation. The plugin for linear real arithmetic uses a similar mechanism of watched variables to keep track of variable bounds [7]. The last checked variable in each clause or a linear constraint is cached. Search for a non-falsified literal or an unassigned rational variable always starts from the last position. Additionally, we use the following heuristics:

- *Variable order.* Yaga uses a generalization of the VSIDS heuristic implementation from MiniSat [10]. Variable score is increased for each variable involved in conflict derivation. Variables of all types (i.e., Boolean and rational variables) are ranked using this heuristic.
- *Restart scheme.* We use a simplified restart scheme from the Glucose solver [2]. The solver maintains an exponential average of glucose level (LBD) of all learned clauses [3] and an exponential LBD average of recently learned clauses. Yaga restarts when the recent LBD average exceeds the global average by some threshold.
- *Clause deletion.* Yaga deletes subsumed learned clauses on restart [5].
- *Clause minimization.* Learned clauses are minimized using self-subsuming resolution introduced in MiniSat [10].
- *Value caching.* Similarly to phase-saving heuristics used in SAT solvers [9], Yaga caches values of decided rational variables [7]. It preferably uses cached values for rational variables. If a cached value is not available, the solver tries to find a small integer or a fraction with a small denominator which is a power of two.
- *Bound caching.* We keep a stack of variable bounds for each rational variable. When the solver backtracks, it lazily removes obsolete bounds from the stack. Bounds computed at a decision level lower than the backtrack level do not have to be recomputed.

2 External Code

The solver uses a custom representation of unbounded rational values from OpenSMT [6], which is itself based on a library written by David Monniaux and uses GMP [1].

3 Availability

The source code repository and more information on the solver is available at

- https://github.com/d3sformal/yaga

References

[1] The GNU multiple precision arithmetic library. https://gmplib.org/. Accessed: 2023-05-13.

[2] Gilles Audemard and Laurent Simon. On the Glucose SAT solver. *International Journal on Artificial Intelligence Tools*, 27:1840001, 02 2018.

[3] Armin Biere. Weaknesses of CDCL solvers. In *Institute Workshop on Theoretical Foundations of SAT Solving*, 2016.

[4] Leonardo de Moura and Dejan Jovanović. A Model-Constructing Satisfiability Calculus. In Roberto Giacobazzi, Josh Berdine, and Isabella Mastroeni, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 1–12, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[5] Niklas Eén and Armin Biere. Effective preprocessing in SAT through variable and clause elimination. In Fahiem Bacchus and Toby Walsh, editors, *Theory and Applications of Satisfiability Testing*, pages 61–75, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[6] Antti E. J. Hyvärinen, Matteo Marescotti, Leonardo Alt, and Natasha Sharygina. OpenSMT2: An SMT Solver for Multi-core and Cloud Computing. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing – SAT 2016*, pages 547–553, Cham, 2016. Springer International Publishing.

[7] Dejan Jovanovic, Clark Barrett, and Leonardo de Moura. The design and implementation of the model constructing satisfiability calculus. In *2013 Formal Methods in Computer-Aided Design*, pages 173–180, 2013.

[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 (IEEE Cat. No.01CH37232)*, pages 530–535, 2001.

[9] Knot Pipatsrisawat and Adnan Darwiche. A lightweight component caching scheme for satisfiability solvers. In João Marques-Silva and Karem A. Sakallah, editors, *Theory and Applications of Satisfiability Testing – SAT 2007*, pages 294–299, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[10] Niklas Sörensson and Niklas Een. Minisat v1.13 - A SAT solver with conflict-clause minimization. *International Conference on Theory and Applications of Satisfiability Testing*, 01 2005.