forl

Mate Soos
Security Research Labs

I. Introduction

This paper presents the defining features of the conflict-driven clause-learning SAT solver *forl*. *forl* aims to be a modern SAT Solver that unifies the ideas present in SatELite [1], PrecoSat [2], glucose [3] and MiniSat [4] with some ideas of the author.

II. Primary features

A. Binary implication graphs

An implication cache mechanism is employed that stores the binary implication graph similarly to stamps [5]. Stamps are also used, as they have been found to aid along with the cache.

B. Clause cleaning

Clauses are cleaned regularly, but neither activities nor glues are used in the cleaning. Instead, the number of times a clause helped to propagate or caused a conflict is used as a measure of the effectiveness. This measure is reset after every cleaning, so clauses have to regularly prove themselves effective to stay in the database.

C. Implicit Clauses

Binary and tertiary clauses are stored and handled implicitly. This greatly eases their subsumption and strengthening. Further, it reduces the cost of creating occurrence lists out of these clauses. Implicit clauses are never cleaned.

D. Statistics

forl gathers large amounts of running statistics. Unfortunately they are not yet used to direct search. However, they can be gathered into MySQL and displayed in a web browser. Importing statistics into the database incurs setup costs and about 10% running cost and so is disabled by default.

E. Time limiting

For average problems inprocessing techniques tend to work well. However, in case of strange problems (such as problems with billions of binary clauses) they sometimes misbehave. This has been solved with more precise time measurements (measuring effort, not actual time) and sometimes complicated time-out checks.

F. Memory usage

Memory usage has been greatly improved with precise tracking of where memory is being used. Although memory leaks are not generally an issue given the programming techniques used, temporary allocation of large data structures was a problem. These issues have been fixed through algorithmic means: e.g. through the use of circular swapping for variable renumbering.

G. Hyper-binary resolution and transitive reduction

On-the-fly hyper-binary resolution [6] and transitive reduction has been implemented in both DFS and BFS probing for both irreducible and reducible binaries. This helps on instances with generally acceptable number of binary clauses. For problems with too many binary clauses, transitive reduction can take too much time. Such cases are detected and transitive reduction is turned off.

H. Certified UNSAT

The DRUP system for certified UNSAT was implemented into *forl*. The current implementation turns on all optimisations except for XOR-manipulation during certificate generation. However, for stamping and implied literal caching to work, binary clauses must never be DRUP-deleted during variable elimination. This trade-off is questionable, as it might considerably slow down proof checking. As such, there are two versions submitted, one with these options turned on, and one with these options turned off.

I. Disjoint component finding

Disjoint components are searched for on a regular basis during solving. These disjoint components are solved with a separate solver instance, renumbering the component's variables such as to minimise the startup time of the subsolver. On certain problems, *forl* can find&solve thousands of disjoint components within a matter of seconds.

III. Miscellaneous optimisations

Hand-rolled memory manager for large clauses, clause offsets instead of pointers, blocking literals, occurrence lists in watchlists, clause abstraction stored in occurrence lists, glue-based and geometric restart selection based on literal polarities, xor detection and manipulation, gate detection and manipulation, variable elimination [1], subsumption, strengthening, on-the-fly subsumption [7], recursive conflict clause minimisation [8] (and automatic disabling in case of bad performance), minimisation with stamps&cache&binary clauses (and automatic disabling in case of bad performance), blocking of long clauses [9], equivalent literal replacement, variable renumbering, literal dominator branching thanks to stamps/cache, dominator probing, polarity caching [10], vivification [11] of long and implicit clauses, watchlist sorting for quasi- prioritised implicit clause propagation, regular cleaning of false literals of all clauses, detection of long trail and consequent restart blocking in case of satisfiable problems, MiniSat-type variable activities, glue-based extra variable activity bumping,

prefetching of watchlists on literal enqueue, optional UIP conflict [12] graph generation, probing (with automatic tuning based on past performance), clause subsumption through irreducible stamps and cache, clause strengthening through reducible&irreducible stamps and cache, precise elimination cost prediction for better elimination order, gradual variable elimination, variable elimination with searching for subsumed&subsuming product clauses.

Acknowledgements

The author would like to thank in no particular order Martin Maurer, Vegard Nossum, Valentin Mayer-Eichberger, George Katsirelos, Karsten Nohl, Luca Melette, Marijn Heule, Vijay Ganesh, Trevor Hansen and Robert Aston for their help.

References

[1] Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In Bacchus, F., Walsh, T., eds.: SAT. Volume 3569 of LNCS., Springer (2005) 61–75

[2] Biere, A.: P{re,i}cosat@sc'09 In: SAT 2009 competitive events booklet. (2009) 41–42

[3] Audemard, G., Simon, L.: GLUCOSE: a solver that predicts learnt clauses quality. In: SAT 2009 competitive events booklet. (2009) 7–8

[4] Eén, N., Sörensson, N.: An extensible SAT-solver. In Giunchiglia, E., Tacchella, A., eds.: SAT. Volume 2919 of LNCS., Springer (2003) 502–518

[5] Heule, M., Järvisalo, M., Biere, A.: Efficient CNF simplification based on binary implication graphs. In Sakallah, K.A., Simon, L., eds.: SAT. Volume 6695 of LNCS., Springer (2011) 201–215

[6] Bacchus, F., Winter, J.: Effective preprocessing with hyper-resolution and equality reduction. In Giunchiglia, E., Tacchella, A., eds.: SAT. Volume 2919 of LNCS., Springer (2003) 341–355

[7] Han, H., Somenzi, F.: On-the-fly clause improvement. [13] 209–222

[8] Sörensson, N., Biere, A.: Minimizing learned clauses. [13] 237–243

[9] Järvisalo, M., Biere, A., Heule, M.: Blocked clause elimination. In Esparza, J., Majumdar, R., eds.: TACAS. Volume 6015 of LNCS., Springer (2010) 129–144

[10] Pipatsrisawat, K., Darwiche, A.: A lightweight component caching scheme for satisfiability solvers. In Marques-Silva, J., Sakallah, K.A., eds.: SAT. Volume 4501 of LNCS., Springer (2007) 294–299

[11] Piette, C., Hamadi, Y., Sais, L.: Vivifying propositional clausal formulae. In Ghallab, M., Spyropoulos, C.D., Fakotakis, N., Avouris, N.M., eds.: ECAI. Volume 178 of Frontiers in Artificial Intelligence and Applications., IOS Press (2008) 525–529

[12] Silva, J.P.M., Sakallah, K.A.: GRASP-a new search algorithm for satisfiability. In: ICCAD'96, IEEE Computer Society (1996) 220–227

[13] Kullmann, O., ed.: Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings. In Kullmann, O., ed.: SAT. Volume 5584 of Lecture Notes in Computer Science., Springer (2009)