IsaSAT and Kissat entering the EDA Challenge 2021

Mathias Fleury
Johannes Kepeler University Linz
Linz, Austria
mathias.fleury@jku.at
Max-Planck-Institut für Informatik, SIC
Saarbrücken, Germany
mathias.fleury@mpi-inf.mpg.de

Abstract—We describe our two SAT solvers that enter the EDA Challenge 2021. While Kissat is virtually identitical to the version submitted to the SAT Competition 2021, our fully verified solver IsaSAT never entered any competition so far and has changed significantly since it was described.

Index Terms—Kissat, IsaSAT, verified SAT solver, SAT solver

I. IsaSAT

IsaSAT [1] is a fully verified SAT solver using the proof assistant Isabelle. Compared to the last described version [1], [2], we generate LLVM code instead of Standard ML (Sect. I-A). To accomodate this change, the solver can also generate unknown (Sect. I-B), although the correctness does not really change. The refinement goes from an nondeterministic abstract CDCL presentation down to deterministic code with heuristic. We added rules to our CDCL formalization to represent simplification with unit clauses (Sect. I-C). We also added new heuristics (Sect. I-D).

A. Code Generation to LLVM

The last step of the refinement in Isabelle to generate IsaSAT is the automatic synthesis of imperative code by the Isabelle Refinement Framework; i.e., lists are replaced by (modifiable) arrays. After the imperative code has be generated, we use the standard code generator to export the imperative code to Haskell, OCaml, Scala, and Standard ML. However, the compiler MLton [3] offers significantly better performance than any other compiler we tried. The code generator that translate the concrete code to executable code is simple enough to be trusted.

We have now changed the synthesis and the code generator to use Lammich's new refinement framework targetting Isabelle/LLVM [4]. The concrete code is now in language very similar to the intermediate representation used by the LLVM compiler [5] (used by, e.g., clang) and the code generation is now trivial pretty-printing. While we did not maintain compatibility with the other code generator,[1] the generated was twice as fast and we could remove some of the memory tricks we had to use to keep the memory representation compact enough: the structure used to watch clauses {uint32_t blit; clause*; bool binary} has size 64 bit and not 128 bit like in Standard ML (where we merged the blit and binary into a single 64-bit integer to reduce the size down to 64 bit, requiring more Isabelle theorems).

Around the verified code, we have a C wrapper that parses the input file and options to simplify experimenting with the solver. One limitation is that the calling convention of pointer is not the same in C code and the verified code itself. It is problematic to pass pointers (like the full model) from the verified code back to the wrapper for printing. We solve this by replacing a verified function doing nothing by a printing function (written in the wrapper). A similar trick is used to print information during the search – the output is optional and deactivated in the wrapper for the competition, however.

B. Correctness

We verify the following theorem on our SAT solver:

Theorem 1 (IsaSAT Correctness): The solver terminates. If the answer is not unknown, then the returned answer (SAT with a model or UNSAT) is correct.

Unlike our previous version, IsaSAT is not complete anymore. Although no proof is provided, incompleteness happens if: (i) more than 2^{64} clauses (the maximum of the type uint64) are learned (including the reasons on the trail); or (ii) the length of the memory region (arena) used to represent clauses is larger than $2^{63} - (2^{16} + 4)$ (the maximum of the type int64 minus the maximum size of the next clause to learn). The latter condition corresponds to keeping more than 2^{47} clauses and is unlikely to trigger on any practical system in any reasonnable time frame. Even if our previous version was complete from the point of view of Isabelle, the code generator assumes that an arbitrary large array can be allocated and accessed using GMP integers. This does do not hold on any compiler we are aware of and hence, the practical correctness of the

I thank Armin Biere for allowing me to submit Kissat and Peter Lammich for helping me port IsaSAT to Isabelle/LLVM. I was supported by Austrian Science Fund (FWF) projects W1255-N23 and S11408-N23, by the LIT AI and LIT Secure and Correct Systems Labs all two funded by the State of Upper Austria.

[1] For technical (Isabelle related) reasons, this would have been very time consumming.

executed code is the same, even if the Isabelle theorem is weaker.

We made two major changes to our refinement approach, but the main difference is that instead of trusting the code translation to Standard ML, now only pretty-printing is trusted.

C. Pragmatic CDCL

Our old version implements a basic version of CDCL with restart and forget. One strong limitation is that we cannot remove clauses that contain a literals know to be unconditionnaly true or remove literals that are unconditionnaly false. This is important for various problems that generate many true literals.

To alleviate we introduced a new calculus called pragmatic CDCL. This calculus extends CDCL with abstract inprocessing transformations. Introducing this new calculus required changing our refinement approach: the calculus reuses our results on CDCL (like termination and completeness) but extended by expressing transformations that cannot be expressed otherwise.

Before any of these transformation is applied, they have to be refined down to concrete executable code. So far only simplification of literals set on level 0 has been refined down to executable code, but more inprocessing can be implemented (in particular subsumption and subsumption are a special case of the rules defined in our pragmatic CDCL).

D. Heuristics

We have implemented several new heuristics that are part of many SAT solvers including target phases and rephasing [6] (excluding random walk to improve models) and the alternation between focus and search mode.

II. Kissat

We basically submitted the exact same version of Kissat [7] also submitted to the SAT Competition 2021 with the new sweeping technique that does not include the (rare) incorrect proofs generated by the default version. For a more detailed list of novelties, please read the system description from the SAT Competition [8].

While our solver IsaSAT offers the highest level of trust that is possible (without checking the proof), the performance is significantly worse than Kissat because no inprocessing is included. Unless you require the extra trust (or cannot afford proof checking), we recommend using another SAT solver, like Kissat.

References

[1] M. Fleury, “Formalization of logical calculi in Isabelle/HOL,” Ph.D. dissertation, Saarland University, Saarbrücken, Germany, 2020. [Online]. Available: https://tel.archives-ouvertes.fr/tel-02963301

[2] ——, “Optimizing a verified SAT solver,” in NASA Formal Methods - 11th International Symposium, NFM 2019, Houston, TX, USA, May 7-9, 2019, Proceedings, ser. Lecture Notes in Computer Science, J. M. Badger and K. Y. Rozier, Eds., vol. 11460. Springer, 2019, pp. 148–165. [Online]. Available: https://doi.org/10.1007/978-3-030-20652-9_10

[3] S. Weeks, “Whole-program compilation in MLton,” in Proceedings of the ACM Workshop on ML, 2006, Portland, Oregon, USA, September 16, 2006. ACM Press, 2006, p. 1.

[4] P. Lammich, “Generating verified LLVM from Isabelle/HOL,” in 10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA, ser. LIPIcs, J. Harrison, J. O’Leary, and A. Tolmach, Eds., vol. 141. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, pp. 22:1–22:19. [Online]. Available: https://doi.org/10.4230/LIPIcs.ITP.2019.22

[5] C. Lattner and V. Adve, “LLVM: A compilation framework for lifelong program analysis & transformation,” in International Symposium on Code Generation and Optimization, 2004. CGO 2004. IEEE, 2004, pp. 75–88.

[6] A. Biere and M. Fleury, “Chasing target phases,” 2020, 11th Workshop on Pragmatics of SAT (POS’20). [Online]. Available: http://www.pragmaticsofsat.org/2020/

[7] A. Biere, K. Fazekas, M. Fleury, and M. Heisinger, “CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020,” in Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions, ser. Department of Computer Science Report Series B, T. Balyo, N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., vol. B-2020-1. University of Helsinki, 2020, pp. 51–53.

[8] ——, “CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2021,” in Proc. of SAT Competition 2021 – Solver and Benchmark Descriptions, 2021, submitted.