Planning and Automated Reasoning – Automated Reasoning
I term, Academic Year 2024-25
Project: Implementation of a solver for the union of the theories of equality, lists, and arrays

Assigned November 12, 2024, to be submitted by 12:00 (noon) February 3, 2025

Assignment

The project consists of the implementation of a prototype solver that determines the satisfiability of a set of literals in the union of the quantifier-free fragments of three theories: equality with free symbols, non-empty possibly cyclic lists, and arrays without extensionality. The heart of the solver is the congruence closure algorithm on DAG's for the satisfiability of a set of equalities and disequalities in the quantifier-free fragment of the theory of equality. The algorithm as described in Sect. 9.3 of the Bradley-Manna textbook should be improved with the following heuristics:

- The forbidden list or forbidden set (see page 388 bottom in reference [2e]);

- A non-arbitrary choice of the representative of the union class in the `UNION` function: pick the one with the largest `ccpar` set (see page 761 top in reference [2f] and page 423 top in reference [2e]);

- A non-recursive version of the `FIND` function; this assumes that the `UNION` function updates the `find` field of all the terms in the class whose representative is not chosen as representative of the union class.

Since these features are heuristics, it is not guaranteed that they are always helpful. Therefore, they can be implemented as options, making it possible to evaluate their impact in practice.

The programming language should be a general-purpose portable programming language that can be compiled on Linux. However, the usage of the Python language is forbidden, because the point of the project is to develop a prototype from scratch, not to assemble existing code.

The program will have an interface (stdin/stdout or a simple GUI) that allows the user to submit an input formula (typically contained in a file) and get the answer.

In order to test the program, use formulas from sources including the following:

1. Examples and exercises from books and papers, beginning with those in the book list of the course and in the references below.

2. Sets of literals obtained from more general formulas (also from books or papers) as follows, e.g.:

   - If the formula contains free predicate symbols, the solver handles them by the transformation explained in class;
   - If the formula is not a conjunction of literals, the solver transforms it into DNF, and then works on each disjunct;
   - If the formula contains defined (constant or function) symbols from theories other than lists and arrays (e.g., arithmetical symbols), the solver handles them by replacing them with free symbols;
   - If the formula contains quantifiers, the solver drops them and treats the variables as free variables.

   Clearly, the last two transformations do not preserve equisatisfiability, and they are intended only to get more inputs for the solver.

3. **Optional:** Synthetic sets of literals obtained from a generator to be implemented in addition to the solver.

4. **Optional:** Benchmarks from `https://smt-lib.org/`: the benchmarks in the `QF-UF` class should fall in the scope of the project. QF stands for quantifier free and UF stands for undefined function symbols. Since all logics/theories in the SMT-LIB repository feature equality, `QF-UF` stands for the quantifier-free fragment of the theory of equality. The benchmarks of SMT-LIB are currently hosted on Zenodo and those for `QF-UF` are at `https://zenodo.org/records/11061097`, but they are also reachable from `https://smt-lib.org/` by clicking on `Benchmarks`. These benchmarks are written in the SMT-LIB language that is far more complex than the logic used in class, books, and papers. Thus, in order to give these inputs to the program, one needs a parser that handles at least the subset of SMT-LIB used in the `QF-UF` benchmarks. Some utilities to handle the SMT-LIB language are available from `https://smt-lib.org/` by clicking on `Utilities`.

The project also requires to write a report (max 6 pages 11pt) presenting

- The implementation, emphasizing major choices (e.g., data structures, the above mentioned heuristics, other heuristics), with comments on their impact (e.g., on ease of implementation, performance) or any other information deemed significant;

- A summary of the results of the experiments in the form of one or more tables or plots, reporting data such as the answer (SAT/UNSAT), the run time, the source of the problems, or other data deemed relevant;

- Some analysis of the experiments, such as comments about performance, impact of features, or any other remarks deemed interesting.

Hand-in

1. A compressed archive (e.g., .tgz[1] or .zip) to be sent to the instructor by e-mail. The archive should contain:

   (a) The source code,

   (b) The input files used in the experiments, each including a comment stating the source of the problem,

   (c) The output files corresponding to the input files,

   (d) A `README` file with instructions on how to execute the program,

   (e) Either a portable Linux executable, or a `README` file which contains also instructions on how to compile/make the program.

   The archive should be named FirstNameLastNameStudentId (i.e., NomeCognomeNumeroMatricola).

2. A double-sided print-out of the report to be put in the instructor's mailbox.

It is not required to include the report in the archive and doing it does not fulfill the requirement of submitting separately the double-sided print-out of the report.

References

1. Textbooks:

   (a) Aaron R. Bradley, Zohar Manna. *The Calculus of Computation. Decision Procedures with Applications to Verification.* Springer, 2007, ISBN 978-3-642-09347-0.

   (b) Daniel Kroening, Ofer Strichman: *Decision Procedures. An Algorithmic Point of View*, Springer, 2008, ISBN: 978-3-540-74104-6.

2. Papers:

   (a) Alessandro Armando, Silvio Ranise, and Michaël Rusinowitch. A rewriting approach to satisfiability procedures. Information and Computation 183(2):140–164, 2003.

   (b) Alessandro Armando, Maria Paola Bonacina, Silvio Ranise, and Stephan Schulz. New results on rewrite-based satisfiability procedures. ACM Transactions on Computational Logic 10(1):129–179, 2009.

---

[1]E.g., `tar czvf - FolderName > FolderName.tgz` compresses and `tar xzpvf FolderName.tgz` decompresses.

(c) Leo Bachmair, Ashish Tiwari and Laurent Vigneron. Abstract congruence closure. Journal of Automated Reasoning 31(2):129–168, 2003.

(d) Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. What's decidable about arrays? In E. Allen Emerson and Kedar S. Namjoshi (Eds.), Proceedings of the Seventh International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI), Lecture Notes in Computer Science 3055:427–442, Springer, 2006.

(e) David L. Detlef, Greg Nelson and James B. Saxe. Simplify: a theorem prover for program checking. Journal of the ACM 52(3):365–473, 2005.

(f) Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. Journal of the ACM 27(4):758–771, 1980.

(g) Dexter Kozen. Complexity of finitely presented algebras. Technical Report TR-76-294, Department of Computer Science, Cornell University, 1976.

(h) Greg Nelson and Derek C. Oppen. Fast decision procedures based on congruence closure. Journal of the ACM 27(2):356–364, 1980.

(i) Robert Nieuwenhuis and Albert Oliveras. Fast congruence closure and extensions. Information and Computation 205:557–580, 2007.

(j) Robert E. Shostak. An algorithm for reasoning about equality. Communications of the ACM 21(7):583–585, 1978.

(k) Aaron Stump, Clark W. Barrett, David L. Dill, and Jeremy Levitt. A decision procedure for an extensional theory of arrays. In Joseph Halpern (Ed.), Proceedings of the 16th IEEE Symposium on Logic in Computer Science, IEEE Computer Society Press, 2001.