Ultimate Automizer with Unsatisfiable Cores*
(Competition Contribution)

Matthias Heizmann, Jürgen Christ, Daniel Dietsch, Jochen Hoenicke,
Markus Lindenmann, Betim Musa, Christian Schilling,
Stefan Wissert, and Andreas Podelski

University of Freiburg, Germany

Abstract. Ultimate Automizer is an automatic software verification tool for C programs. This tool is a prototype implementation of an automata-theoretic approach that allows a modular verification of programs. Furthermore, this is the first implementation of a novel interpolation technique where interpolants are not obtained from an interpolating theorem prover but from a combination of a live variable analysis, interprocedural predicate transformers and unsatisfiable cores.

1 Verification Approach

Ultimate Automizer verifies a C program by first executing several program transformations and then performing an interpolation-based variant of trace abstraction [4]. As a first step, we translate the C program into a Boogie [6] program. The heap of the system is modeled via arrays in this Boogie program [7]. Next, the Boogie program is translated into an interprocedural control flow graph [9]. As an optimization, we do not label the edges with single program statements but with loop free code blocks of the program [11]. Our verification algorithm then performs the following steps iteratively:

1. We take a sequence of statements π that leads from the start of the main procedure to an error location and analyze its correctness (resp. feasibility). In this analysis an SMT solver is used.
2. We consider this sequence of statements as a standalone program $\mathcal{P}_\pi$ and compute a correctness proof for $\mathcal{P}_\pi$ in form of a Hoare annotation.
3. We find a larger program $\widehat{\mathcal{P}}_\pi$ that has the same correctness proof [4].
4. We consider the preceding step as a semantical decomposition of the original program $\mathcal{P}$ into one part $\widehat{\mathcal{P}}_\pi$ whose correctness is already proven and one remaining part $\mathcal{P}_{\mathsf{rest}} := \mathcal{P} \backslash \widehat{\mathcal{P}}_\pi$, on which we continue. The programs $\mathcal{P}, \widehat{\mathcal{P}}_\pi, \mathcal{P}_{\mathsf{rest}}$ are represented by automata. This allows us to compute and represent the remaining part of the program $\mathcal{P}_{\mathsf{rest}}$ (the part for which correctness was not yet proven). Furthermore, this automata-theoretic representation allows us to apply minimization [10] to represent the programs $\mathcal{P}, \widehat{\mathcal{P}}_\pi, \mathcal{P}_{\mathsf{rest}}$ efficiently.

* This work is supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR14 AVACS)

E. Ábrahám and K. Havelund (Eds.): TACAS 2014, LNCS 8413, pp. 418–420, 2014.

Our previous competition candidate [2] followed a similar approach in which the above mentioned Hoare annotation was computed by an interpolating SMT solver via Craig interpolation. Computation of Craig interpolants is known to be difficult, especially for the theory of arrays. This competition candidate follows a novel approach [8] to obtain a Hoare annotation for a sequence of statements. The predicates that represent the Hoare annotation are obtained using interprocedural predicate transformers. The arguments of these predicate transformers are not the statements of the sequence but generalized statements that are obtained from a live variable analysis and from unsatisfiable cores of the feasiblity analysis.

2 Software Architecture

Ultimate Automizer is one toolchain of the software analysis framework Ultimate which is implemented in Java. Ultimate offers data structures for different representations of a program, plugins which analyze or transform a program, and an interface for the communication with SMT-LIBv2 compatible theorem provers. For parsing C programs, we use the C parser of the Eclipse CDT project[1]. The operations on nested word automata are implemented in the Ultimate Automata Library. Our SMT queries can be answered by any SMT-LIBv2 compatible solver that supports quantifiers and the theory of arrays.

3 Discussion of Approach

Currently we model primitive data types (`int`, `float`,...) as integers $\mathbb{Z}$ or real numbers $\mathbb{R}$. We report *unknown* whenever we find a potential counterexample whose infeasibility cannot be shown because of this imprecision.

The main flaw of our implementation is the translation from C to Boogie. We failed to finish this translation in time and our submitted competition candidate is unable to verify programs that contain pointers or arrays.

4 Tool Setup and Configuration

Our competition candidate assumes that version 4.3.2.ff265c6c6ccf of the SMT solver Z3[2] is installed and that the directory of the Z3 binary is part of the PATH variable. Our competition candidate is included in a command-line version of Ultimate Automizer that can be downloaded from the following website:

`https://ultimate.informatik.uni-freiburg.de/automizer/`

The zip archive in which Ultimate Automizer is shipped contains the Python script `automizerSV-COMP.py` which wraps input and output for the SV-COMP.

[1] `https://www.eclipse.org/cdt/`
[2] `https://z3.codeplex.com/`

Using the following command, the C program `fnord.c` is verified with respect to the property file `prop.prp` and an error path is written to the file `errPath.txt`.

```
python AutomizerSvcomp.py  prop.prp  fnord.c  errPath.txt
```

5 Software Project and Contributors

Our software analysis framework Ultimate was started as a bachelor thesis [1]. In the last years, many students contributed plugins or improved the framework itself. A list of all developers is available on our website. An instance of Ultimate is running on our web server and is available via a web interface.

6 Demonstration Category Termination

We also participated in the demonstration category on termination with Ultimate Büchi Automizer which is our tool for termination analysis. The underlying approach is based on Büchi automata and has not been published yet. As a subroutine the tool Ultimate Lasso Ranker [3,5] is used. We thank Jan Leike and Alexander Nutz for their contributions to our termination analysis.

References

1. Dietsch, D.: STALIN: A plugin-based modular framework for program analysis. Bachelor Thesis, Albert-Ludwigs-Universität, Freiburg, Germany (2008)
2. Heizmann, M., et al.: Ultimate automizer with SMTInterpol. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 641–643. Springer, Heidelberg (2013)
3. Heizmann, M., Hoenicke, J., Leike, J., Podelski, A.: Linear ranking for linear lasso programs. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 365–380. Springer, Heidelberg (2013)
4. Heizmann, M., Hoenicke, J., Podelski, A.: Software model checking for people who love automata. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 36–52. Springer, Heidelberg (2013)
5. Leike, J.: Ranking function synthesis for linear lasso programs. Master's thesis, University of Freiburg, Germany (2013)
6. Leino, K.R.M.: This is Boogie 2. Manuscript working draft, Microsoft Research, Redmond, WA, USA (June 2008), `http://research.microsoft.com/en-us/um/people/leino/papers/krml178.pdf`
7. Lindenmann, M.: A simple but sufficient memory model for ultimate. Master's thesis, University of Freiburg, Germany (2012)
8. Musa, B.: Trace abstraction with unsatisfiable cores. Bachelor's thesis, University of Freiburg, Germany (2013)
9. Reps, T.W., Horwitz, S., Sagiv, S.: Precise interprocedural dataflow analysis via graph reachability. In: POPL 1995, pp. 49–61. ACM (1995)
10. Schilling, C.: Minimization of nested word automata. Master's thesis, University of Freiburg, Germany (2013)
11. Wissert, S.: Adaptive block encoding for recursive control flow graphs. Master's thesis, University of Freiburg, Germany (2013)