Ultimate Automizer with SMTInterpol*
(Competition Contribution)

Matthias Heizmann, Jürgen Christ, Daniel Dietsch, Evren Ermis, Jochen Hoenicke, Markus Lindenmann, Alexander Nutz, Christian Schilling, and Andreas Podelski

University of Freiburg, Germany

Abstract. Ultimate Automizer is an automatic software verification tool for C programs. This tool is the first implementation of *trace abstraction*, which is an automata-theoretic approach to software verification. The implemented algorithm uses *nested interpolants* in its interprocedural program analysis. The interpolating SMT solver SMTInterpol is used to compute Craig interpolants.

1 Verification Approach

UltimateAutomizer verifies a C program by first executing several program transformations and then performing an interpolation based variant of trace abstraction [5].

As a first step we translate the C program into a Boogie [7] program. Next, the Boogie program is translated into an interprocedural control flow graph [8]. As an optimization we do not label the edges with single program statements but with loop free code blocks of the program [2].

In our algorithm, the program is represented by an automaton which accepts all error traces of the program. An error trace is a labeling of an initial path in the control flow graph which leads to the error location. If all error traces are infeasible with respect to the semantics of the programming language, the program is correct.

We use the CEGAR algorithm depicted below. Our abstraction is a nested word automaton [1] $\mathcal{A}_{\mathsf{error}}$ which accepts only error traces of the program. We iteratively subtract from our abstraction $\mathcal{A}_{\mathsf{error}}$ the language of an automaton $\mathcal{A}_{\mathcal{I}}$ which accepts only infeasible traces. The algorithm terminates if the language of $\mathcal{A}_{\mathsf{error}}$ is empty or a feasible error trace π was found.

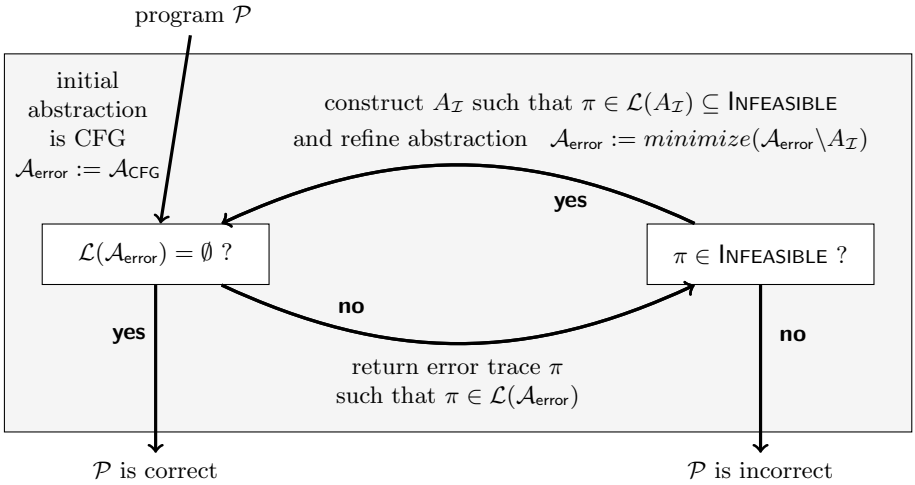
The automaton $\mathcal{A}_{\mathcal{I}}$ which accepts only infeasible traces is constructed as follows. If the error trace π is infeasible we consider π as a straight-line program which has a Hoare annotation $\mathcal{I}$ where the initial location is labeled with the *true* predicate and the final state assertion is the *false* predicate. We call such

* 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).

N. Piterman and S. Smolka (Eds.): TACAS 2013, LNCS 7795, pp. 641–643, 2013.
© Springer-Verlag Berlin Heidelberg 2013

a Hoare annotation $\mathcal{I}$ a sequence of nested interpolants [6] for π. We compute a sequence of nested interpolants by recursively computing sequences of Craig interpolants.

After subtracting the language of the interpolant automaton $\mathcal{A}_{\mathcal{I}}$ from our abstraction $\mathcal{A}_{\mathsf{error}}$, we apply a minimization for nested word automata which preserves the language of $\mathcal{A}_{\mathsf{error}}$, but reduces the number of states significantly in many cases.



2 Software Architecture

Ultimate Automizer is one toolchain of the software analysis framework Ultimate[1] 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[2]. The operations on nested word automata are implemented in the Ultimate Automata Library. As interpolating SMT solver we use SMTInterpol[3] [3].

3 Discussion of Approach

Conceptually, our approach is applicable to each class of programs whose semantics can be defined via SMT formulas. However, the current implementation of Ultimate Automizer supports only sequential programs and does neither support arrays, pointers, nor bitvector operations.

[1] `http://ultimate.informatik.uni-freiburg.de/`
[2] `http://www.eclipse.org/cdt/`
[3] `http://ultimate.informatik.uni-freiburg.de/smtinterpol/`

4 Tool Setup and Configuration

Our competition candidate is a version of Ultimate with a command line user interface that contains a version of SMTInterpol and can be downloaded from the following website:

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

The zip archive in which Ultimate Automizer is shipped contains the bash script `automizerSV-COMP.sh` which calls Ultimate with all parameters that are necessary to verify C programs using the Ultimate Automizer toolchain. In order to verify the C program `fnord.c`, use the directory where you extracted the zip archive as your working directory and execute the following command:

```
automizerSV-COMP.sh  fnord.c
```

5 Software Project and Contributors

Our software analysis framework Ultimate was started as a bachelor thesis [4]. In the last years, many students contributed plugins or improved the framework itself. Soon we will release two user interfaces for Ultimate, a web interface and a plugin for the Eclipse CDT project. In both user interfaces you can also use the UltimateAutomizer toolchain to verify C programs.

The Authors thank Alex Saukh and Stefan Wissert for their contributions to the plugin which translates C programs to Boogie programs. Furthermore the Authors thank all developers that contributed to Ultimate, to the Ultimate Automata Library, or to SMTInterpol.

References

1. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM 56(3), 16:1–16:43 (2009)
2. Beyer, D., Cimatti, A., Griggio, A., Keremoglu, M.E., Sebastiani, R.: Software model checking via large-block encoding. In: FMCAD, pp. 25–32. IEEE (2009)
3. Christ, J., Hoenicke, J., Nutz, A.: Proof Tree Preserving Interpolation. In: Piterman, N., Smolka, S. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 123–137. Springer, Heidelberg (2013)
4. Dietsch, D.: STALIN: A plugin-based modular framework for program analysis. Bachelor Thesis, Albert-Ludwigs-Universität, Freiburg, Germany (2008)
5. Heizmann, M., Hoenicke, J., Podelski, A.: Refinement of Trace Abstraction. In: Palsberg, J., Su, Z. (eds.) SAS 2009. LNCS, vol. 5673, pp. 69–85. Springer, Heidelberg (2009)
6. Heizmann, M., Hoenicke, J., Podelski, A.: Nested interpolants. In: Hermenegildo, M.V., Palsberg, J. (eds.) POPL, pp. 471–482. ACM (2010)
7. 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`
8. Reps, T.W., Horwitz, S., Sagiv, S.: Precise interprocedural dataflow analysis via graph reachability. In: POPL 1995, pp. 49–61. ACM (1995)