AProVE: Termination and Memory Safety of C Programs*
(Competition Contribution)

T. Ströder, C. Aschermann, F. Frohn, J. Hensel, and J. Giesl

RWTH Aachen University, Germany

Abstract. AProVE is a system for automatic termination and complexity proofs of C, Java, Haskell, Prolog, and term rewrite systems. The particular strength of AProVE when analyzing C is its capability to reason about pointer arithmetic combined with direct memory accesses (as, e.g., in standard implementations of string algorithms). As a prerequisite for termination, AProVE also proves memory safety of C programs.

1 Verification Approach and Software Architecture

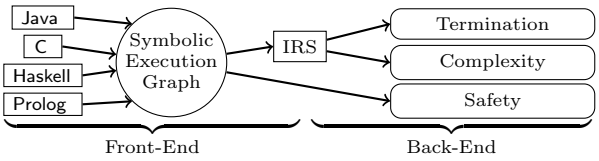
To analyze programs with explicit pointer arithmetic, one has to handle the interplay between addresses and the values they point to. AProVE uses an approach based on symbolic execution and abstraction to transform the input program into a *symbolic execution graph* that over-approximates all possible program runs. Language-specific features (such as pointer arithmetic in C) are handled when generating this graph. The nodes of the symbolic execution graph are abstract states that represent sets of actual program states, and paths in the graph correspond to evaluations in the program. To keep the graph finite, we use *abstraction* to replace several states at the same program location by a more general new state. To formalize abstract states, we introduce a novel abstract domain that can track allocated memory in detail. An important advantage of our domain is that although it is based on separation logic, standard integer SMT solving can be used for all reasoning needed in our approach. Thus, the rules for symbolic execution and generalization of states can easily be automated.

In C, a violation of *memory safety* (i.e., accessing non-allocated memory) leads to undefined behavior, which may also include non-termination. Therefore, to prove termination of C programs with low-level memory access, one must also ensure memory safety. Hence, during the construction of the symbolic execution graph, we also prove memory safety of our input program. In a similar way, one could also prove other safety properties by checking that there is no path from initial states to "unsafe" states in the graph.

After verifying memory safety, the graph is automatically transformed into an integer rewrite system (IRS) whose termination is analyzed afterwards. In this way, the same termination techniques in the *back-end* of AProVE are used for termination analysis of different programming languages in the *front-end*.

* Supported by DFG grant GI 274/6-1 and Research Training Group 1298 (*AlgoSyn*).

A graphical overview of AProVE's architecture is shown on the right. Details on our approach for analyzing C programs can be found in [9]. In [7], we explain the use of AProVE for other programming languages and give references to our corresponding papers on the underlying theory.



2 Strengths and Weaknesses

The strength of our approach for C is that it handles algorithms where the control flow depends on explicit pointer arithmetic and on detailed information about the contents of addresses, whereas most other tools fail for such algorithms. Moreover, in contrast to AProVE, most other termination provers ignore the problem of memory safety and just prove termination under the *assumption* that the program is memory safe. The success of AProVE at the annual international *Termination Competition*[1] shows that our rewriting-based approach is well suited for termination analysis of real-world programming languages. Here, AProVE won almost all categories related to termination of Java, Haskell, Prolog, and to termination or innermost runtime complexity of rewriting. Moreover, AProVE was the most powerful tool for termination analysis of C (the competition had such a category for the first time in 2014). At *SV-COMP*, AProVE already participated very successfully in 2014 when the competition featured a demonstration category for termination of C programs for the first time. This year, AProVE won the termination category of *SV-COMP* by proving termination for 305 of the 395 programs in this category (44 of the remaining programs are non-terminating, thus AProVE was successful on approx. 87% of the terminating ones).

On the other hand, since AProVE constructs symbolic execution graphs to prove memory safety and to infer suitable invariants needed for termination proofs, its runtime is often higher than that of other tools. Moreover, since symbolic execution graphs over-approximate the set of actual program runs, AProVE currently cannot *disprove* termination or memory safety.[2] A further weakness is that we only handle algorithms with integers and pointers, but no `struct` types yet, and that we assume integers to be unbounded. This is why AProVE currently only participates in the termination category, since this category assumes unbounded integers and the examples in this category do not contain `struct`s. Finally, as our approach is targeted toward termination, up to now we did not implement support for other forms of safety besides memory safety.

3 Setup and Configuration

AProVE is developed by the "*Programming Languages and Verification*" group

[1] `http://www.termination-portal.org/wiki/Termination_Competition`

[2] AProVE already proves non-termination of term rewriting and Java. We are currently working on adapting these techniques to the abstract domain used for C programs.

led by Jürgen Giesl at RWTH Aachen University. AProVE's main website is [1]. Here, AProVE can be downloaded as a command-line tool or as a plug-in for the popular Eclipse software development environment [5]. In this way, AProVE can already be applied during program construction. Moreover, AProVE can also be accessed directly via a web interface. The website [1] also contains a list of external tools used by AProVE and a list of present and past contributors.

The particular version for analyzing C programs according to the *SV-COMP* format can be downloaded from the following URL. In this version, we disabled the check for memory safety, since it was agreed that only memory safe programs will be included in the termination category of *SV-COMP*.

`http://aprove.informatik.rwth-aachen.de/eval/Pointer/AProVE.zip`

All files from this archive have to be extracted into one folder. AProVE is implemented in Java and needs a Java 7 Runtime Environment. To avoid handling the intricacies of C, we analyze programs in the platform-independent intermediate representation of the LLVM compilation framework [8] and AProVE requires the Clang compiler Version 2.9 [2] to translate C sources to LLVM. To solve the arising search problems in the back-end, AProVE needs the satisfiability checkers Z3 [3], Yices [4], and MiniSAT [6]. Moreover, extending the path environment is necessary so that AProVE can find the corresponding programs.

AProVE participated in the category "*Termination*". It can be invoked for C files using the following call pattern. Here, `<problemFile>` is the C file to be analyzed for termination of the call `main()`, while `<outputFile>` is a file where AProVE should store its proof (or proof attempt).

```
./AProVE.sh <problemFile> <outputFile>
```

AProVE prints TRUE on the standard output if it can prove termination. Otherwise it prints UNKNOWN. As mentioned, currently, AProVE is not able to disprove termination for C programs, so AProVE does not print FALSE.

References

1. AProVE. `http://aprove.informatik.rwth-aachen.de/`.
2. Clang. `http://clang.llvm.org`.
3. L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *TACAS '08*, pages 337–340.
4. B. Dutertre and L. de Moura. The Yices SMT solver, 2006. Tool paper at `http://yices.csl.sri.com/tool-paper.pdf`.
5. Eclipse. `http://www.eclipse.org/`.
6. N. Eén and N. Sörensson. An extensible SAT-solver. In *SAT '03*, pages 502–518.
7. J. Giesl, M. Brockschmidt, F. Emmes, F. Frohn, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. Proving termination of programs automatically with AProVE. In *IJCAR '14*, pages 184–191.
8. C. Lattner and V. S. Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *CGO '04*, pages 55–88.
9. T. Ströder, J. Giesl, M. Brockschmidt, F. Frohn, C. Fuhs, J. Hensel, and P. Schneider-Kamp. Proving termination and memory safety for programs with pointer arithmetic. In *IJCAR '14*, pages 208–223.