SMACK+Corral: A Modular Verifier*

(Competition Contribution)

Arvind Haran[1], Montgomery Carter[1], Michael Emmi[2],
Akash Lal[3], Shaz Qadeer[3], and Zvonimir Rakamarić[1]

[1] School of Computing, University of Utah, USA
zvonimir@cs.utah.edu
[2] IMDEA Software Institute, Spain
michael.emmi@imdea.org
[3] Microsoft Research, India & USA
akashl@microsoft.com

Abstract. SMACK and Corral are two components of a modular toolchain for verifying C programs. Together they exploit state-of-the-art compiler technologies and theorem provers to simplify and dispatch verification conditions.

1 Verification Approach

SMACK [3] is a translator from the LLVM compiler's intermediate representation (IR) into the Boogie intermediate verification language (IVL) [1]. Sourcing LLVM exploits a number of frontends, optimizations, and analyses. Targeting Boogie exploits a canonical platform which simplifies verifier implementations.

Corral [2] is a verifier for the Boogie IVL which views programs as control flow over any SMT-encodable expression language. Corral delegates semantic reasoning to SMT solvers, and in minimizing syntactic program assumptions, it is compatible with any theory supported by the underlying solvers.

SMACK+Corral leverages multiple theories to encode various C-language features. We can model memory in array theory, non-linear operations with uninterpreted functions, fixed-width words in bitvector theory, and arbitrary-length words in linear arithmetic. Though we make no attempt to generate inductive invariants, we can use any invariant generator as a pre-pass; if proved sound, the resulting invariants are injected into the program as assumptions which help Corral narrow its search.

2 Software Architecture

Figure 1 depicts the SMACK+Corral architecture. We leverage the LLVM[1] compiler's Clang C language family frontend to generate LLVM IR, an assembly-like language in *single static assignment* (SSA) form targeted by frontends for a diverse spectrum of languages (e.g., Java, JavaScript, Haskell, Erlang, Fortran) which is a convenient

* Partially supported by NSF award CCF 1346756 and a Microsoft Research SEIF award.

[1] http://llvm.org and http://clang.llvm.org

C. Baier and C. Tinelli (Eds.): TACAS 2015, LNCS 9035, pp. 451–454, 2015.
DOI: 10.1007/978-3-662-46681-0_42

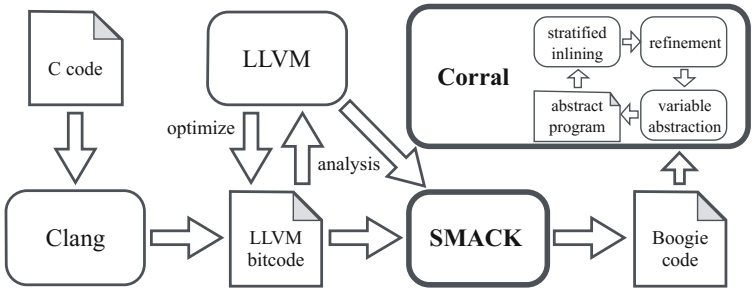

Fig. 1. The SMACK+Corral architecture

representation for code optimization. We then exploit LLVM to perform several code optimizations including control-flow graph simplification, constant propagation, and memory-to-register promotion. Collectively these optimizations can substantially simplify the source C program with fewer control locations and memory operations.

SMACK translates from the LLVM IR to the Boogie *intermediate verification language* (IVL). The Boogie IVL is a simple imperative language with well-defined, clean, and mathematically-focused semantics which is a convenient representation for software verifiers. Internally, SMACK leverages LLVM pointer-aliasing analyses to construct effective encodings of pointer and memory operations into Boogie, e.g., to avoid encoding program memory as one single array expression, which would be difficult for back-end verifiers to reason about.

Corral attempts to prove reachability of assertion violations in the Boogie program generated by SMACK *lazily*, in a goal-directed manner, to reduce pressure on the underlying theorem prover. Corral abstracts the input program via *variable abstraction*, attempting to identify a minimal set of global variables impacting the verification condition, and *stratified inlining*, attempting to identify a minimal unrolling of program loops and recursion impacting the verification condition. When necessary, Corral refines these abstractions by tracking additional global variables and further unrolling.

3 Strengths and Weaknesses of the Approach

Speaking generally, the main incentives of our approach are modularity and the exploitation of scalable technologies. Sourcing LLVM IR exploits a rapidly-growing frontier of LLVM frontends, encompassing a diverse set of languages including C/C++, Java, Haskell, Erlang, Python, Ruby, Ada, and Fortran. In addition, we benefit from code simplifications made by LLVM's optimizer, including constant propagation and CFG simplification, as well as readily-available analyses, including LLVM's pointer analyses. SMACK's translation to Boogie IVL exploits a canonical platform which simplifies the implementation of verifiers like Corral due to Boogie's minimal syntax and mathematically-focused expression language. Finally, by cleverly exploiting the power of efficient satisfiability modulo theories (SMT) solvers, Corral is able to scale up to complex verification queries on large programs. The general weaknesses of our

approach are currently the limited support for proving programs correct, and the limited support for certain C-language features such as floating-point and bitwise operations.

4 Tool Setup and Configuration

Our SV-COMP 2015 submission[2] contains a prebuilt Linux binary without external dependencies, and is run by invoking the top-level script `smack-svcomp.sh`. The following command line options should be provided for SV-COMP benchmarks:

- `--outputdir` specifies a path where temporary files are generated;
- `--errorwitness` specifies the file name for an output error witness;
- `--m64` must be set on 64-bit benchmarks, such as Device Drivers Linux 64-bit.

For example, SMACK is invoked on a C benchmark file `b.c` by running

```
smack-svcomp.sh b.c --outputdir /scratch --errorwitness /tmp/w.xml
```

the result of which is either `TRUE`, `UNKNOWN`, or `FALSE(REACH)`, in which case an error witness is written to `/tmp/w.xml`.

SV-COMP Categories: Arrays, Control Flow and Integer Variables, Device Drivers Linux 64-bit, Heap Manipulation/Dynamic Data Structures, Recursive, and Simple.

Note: We preprocess SV-COMP benchmarks by removing `#N-source-lines`, `#pragma`, and `#line`, since tokenization breaks otherwise. The SV-COMP error witness checker must do the same for token numbers to match. We provide a simple Python script called `replacer.py` with our binary to perform this transformation.

5 Software Project and Contributors

SMACK is an MIT-licensed open-source project hosted by GitHub[3] developed and maintained by Michael Emmi of the IMDEA Software Institute and Zvonimir Rakamarić of the University of Utah, with additional contributions from Montgomery Carter, Arvind Haran, and Pantazis Deligiannis. SMACK is also hosted by Microsoft's rise4fun[4] website, which allows installation-free use. Corral is an Apache 2.0-licensed open-source project hosted by CodePlex[5] developed and maintained by Akash Lal and Shaz Qadeer of Microsoft Research. Corral is distributed with Microsoft's Static Driver Verifier, included in the Windows Driver Development Kit. Both SMACK and Corral are components of the Q modular verification-technology ecosystem[6].

[2] `http://soarlab.org/smack/smack-corral.tar.gz`
[3] `https://github.com/smackers/smack`
[4] `http://rise4fun.com/SMACK`
[5] `http://corral.codeplex.com`
[6] `http://research.microsoft.com/en-us/projects/verifierq`

References

1. DeLine, R., Leino, K.R.M.: BoogiePL: A typed procedural language for checking object-oriented programs. Technical Report MSR-TR-2005-70, Microsoft Research (2005)
2. Lal, A., Qadeer, S., Lahiri, S.K.: A solver for reachability modulo theories. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 427–443. Springer, Heidelberg (2012)
3. Rakamarić, Z., Emmi, M.: SMACK: Decoupling source language details from verifier implementations. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 106–113. Springer, Heidelberg (2014)