LCTD: Tests-Guided Proofs for C Programs on LLVM

(Competition Contribution)

Olli Saarikivi* and Keijo Heljanko

Helsinki Institute for Information Technology HIIT,
Department of Computer Science,
Aalto University, School of Science,
PO Box 15400, FI-00076 Aalto, Finland
{olli.saarikivi, keijo.heljanko}@aalto.fi

Abstract. LCTD is an open source verification tool for C programs. It uses the LLVM compiler framework to instrument programs for verification with the DASH algorithm. LCTD has been submitted to the BitVectorsReach category of SV-COMP 2016.

1 Verification Approach

The DASH algorithm by Beckman et al. [1] combines dynamic symbolic execution (DSE) [2] with CEGAR. DASH attempts to generate tests based on counterexamples found in the abstraction. When test generation fails the abstraction is refined via a splitting operation on the abstract regions to remove the counterexample. The tests can be seen as an underapproximation of the reachable states of the program under test, which DASH tries to expand to include an error. The abstraction on the other hand is an overapproximation which, if error free, also proves the program under test to be so.

The flowchart in Fig. 1 provides a high-level overview of the DASH algorithm. DASH implements a modified CEGAR loop, where instead of directly checking whether a counterexample is spurious, DSE is used to generate a test that follows the path to the error in the abstraction at least one step more than in previously executed tests. When test generation fails abstraction refinement is performed to eliminate the path from the abstraction.

We have implemented the DASH algorithm as a modification to the Lime Concolic Tester (LCT) [3], which is an open source dynamic symbolic execution tool for C and Java programs. Our tool LCTD extends the LLVM based C support in LCT. For a detailed description of LCTD see [4].

2 Software Architecture

LCTD consists of two main parts:

* Corresponding author

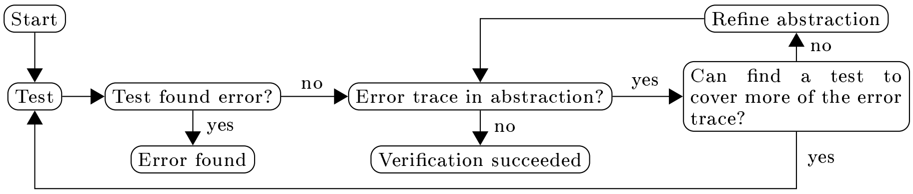

Fig. 1. Flowchart for the DASH algorithm

- An instrumented version of the program to verify, which implements test execution and tracking, and constraint solving.
- A server component which maintains and refines the abstraction.

The target program is instrumented with a transformation pass in the LLVM compiler framework, which adds for all LLVM IR instructions calls to counterparts in a runtime library. These calls allow the runtime to track the execution and provide concrete values for calls to the `__VERIFIER_nondet_*` functions.

At startup the instrumented program connects to the server component for instructions. For test executions it receives a set of concrete inputs, which are used to execute the program. During execution tracking information will be sent to the server, which follows the execution's progress in the abstraction. For solving new input values the server sends a set of concrete inputs and a constraint to be solved at a specific point in the execution, which corresponds to generating a test that visits a desired abstract region. Constraints are solved using the Z3 4.3.2 SMT solver.

The server component initializes the abstraction to the control flow graph of the target program. It waits for the instrumented program to connect, which it then uses for executing tests and solving constraints.

3 Strengths and Weaknesses of the Approach

LCTD implements a bit-precise translation from LLVM IR instructions into bitvector logic making the tool very precise. The usage of a modern SMT solver allows LCTD to perform well on programs with complex bitwise logic.

LCTD leverages LLVM's optimization passes as a preprocessing step. This allows it to produce a simpler version of the program which often omits lots of inessential code and thus verify an optimized LLVM representation of the program.

One of the current challenges is that programs that rely heavily on control flow or complex loops can result in LCTD splitting the abstraction along increasingly deep paths, which results in very large region predicates that are slow to solve. Other weaknesses are limited support for floating point operations, pointer arithmetic and recursive functions.

4 Tool Setup and Configuration

LCTD and its benchmark definition XML can be downloaded from:
`http://users.cse.aalto.fi/osaariki/lctd-svcomp/`

The BenchExec script is available at:
`https://github.com/OlliSaarikivi/benchexec/blob/master/benchexec/tools/lctd.py`

The version to use is "lctd-1.1.1-svcomp". To install the tool:

- Install a Java VM version 1.7.0_79 or newer. LCTD has been tested with Java 1.7.0_79 OpenJDK (IcedTea 2.5.6).
- Add the "bin/" folder inside the root directory of the tool archive to PATH.

Invoking the command "lctdsvcomp <path-to-target.c>" instruments the program and starts the verification process. Once finished it will report TRUE, FALSE or UNKNOWN and in the case of FALSE provides a path to and printout of the verification witness file. LCTD does not require any parameters apart from a path to the source code of the program to verify.

Participation statement: LCTD participates in the BitVectorsReach subcategory and opts out of all other categories.

We do not participate in Overflows, the other bit vector sub-category, as LCTD currently only supports code reachability properties. Other categories were excluded mainly due to a variety of language support issues.

5 Software Project and Contributors

The main developer of LCTD is Olli Saarikivi. The tool was developed by Olli Saarikivi for a Master's Thesis under the supervision of Keijo Heljanko. LCTD is based on the LCT-C tool developed in the Lime project (`http://www.tcs.hut.fi/Software/lime/`).

LCTD is licensed under the MIT license.

References

1. Beckman, N.E., Nori, A.V., Rajamani, S.K., Simmons, R.J.: Proofs from tests. In: Ryder, B.G., Zeller, A. (eds.) Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA). pp. 3–14. ACM (2008)
2. Godefroid, P., Klarlund, N., Sen, K.: DART: Directed automated random testing. In: Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation (PLDI). pp. 213–223. ACM (2005)
3. Kähkönen, K., Launiainen, T., Saarikivi, O., Kauttio, J., Heljanko, K., Niemelä, I.: LCT: An open source concolic testing tool for Java programs. In: Proceedings of the 6th Workshop on Bytecode Semantics, Verification, Analysis and Transformation (BYTECODE). pp. 75–80 (2011)
4. Saarikivi, O., Heljanko, K.: LCTD: Test-guided proofs for C programs on LLVM. Journal of Logical and Algebraic Methods in Programming, NWPT 2013 special issue. (2015)