Project Proposal
CS639A - Fall 2020
Baldip Singh Bijlani, Yugesh Ajit Kothari

Introduction

WebAssembly[1] is a safe, portable, low-level bytecode language and stack-based virtual machine designed for the increasingly sophisticated domain of Web applications. It offers compact representation, efficient validation and compilation, and low to no-overhead execution. It is being developed by a W3C community group.
WebAssembly allows developers to take their native C/C++ code to the browser, where it can be run with near-native performance by the end user.
As of this writing, all major browsers support executing WebAssembly (.wasm binaries). As adoption of WebAssembly via native C/C++ code increases, it becomes interesting to verify program properties at different levels of hierarchy - verifying both native C/C++ code intended to be compiled to wasm and the wasm byte-code itself. An open source compiler toolchain called Emscripten[2] has comprehensive and portable support for C/C++ standard libraries, WebGL APIs, and can be used to write C/C++ code for the web.
In this project we propose to explore the verification of native C/C++ code written using the emscripten SDK. We propose the use of Angelic Verification[3] and model library calls in open programs intended to be used on the Web.

Project Objectives

Our goal is to explore the use of Angelic Verification using the Smack toolchain[4,5] (smack translator + Corral) to verify C/C++ programs which have emscripten library calls and are intended to be compiled as wasm. We wish to answer the following research question : Can we leverage AV for library modelling of open programs, here native C/C++ source using emscripten SDK, in a way which minimises false positives due to a lack of models for library calls in emscripten's SDK? Concretely, our attempt is to meet the following objectives :

1. Identify a set of benchmark programs that capture a related subset of library features common to programs written in native C/C++ with intent to compile and use them as WebAssembly binaries.
2. Identify false positives in the benchmark programs chosen above, when verified using corral without models for library functions.
3. Develop a vocabulary, and angelic assertions for the benchmark to leverage Angelic Verification using Corral (an angelic verifier is parametric in a vocabulary and angelic assertions).
4. Use AV to reduce false positives as compared to verification with only corral (without models of the library calls).

Deliverables

1. Baseline experiments on benchmark using corral (without models, stubs, angelic verification,etc. for library calls) to see verification results without knowledge of library calls. This includes the frequency of false positives and bugs detected.
2. Comparison of baseline experiments with experiments that use models for library calls.
3. Comparison of baseline experiments with experiments using models for library calls inferred using Angelic verification.
4. Potential proof-of-concept for leveraging AV to verify native C/C++ code written for the web.

Technologies

1. Emscripten:
    a. An open source compiler toolchain that compiles C and C++ to WebAssembly using LLVM and Binaryen. Emscripten output can run on the Web, in Node.js, and in wasm runtimes.
2. Boogie[6]:
    a. Boogie is an intermediate verification language, intended as a layer on which to build program verifiers for other languages.
3. Corral:
    a. Corral is a whole-program analysis tool for boogie programs. Corral uses goal-directed symbolic search techniques to find assertion violations. It leverages the powerful theorem prover Z3.
4. Smack:
    a. SMACK is a translator from the LLVM compiler's popular intermediate representation (IR) into the Boogie intermediate verification language (IVL).
5. Angelic Verification:
    a. It is a technique for verification of open programs, where a verifier is constrained to report warnings only when no acceptable environment specification exists to prove the assertion. It is parametric in a vocabulary and a set of angelic assertions that allows a user to configure the tool.

Milestones

6th November (Milestone 1):

- Setup tools and scripts for complete verification pipeline.
- Identify and prepare benchmarks.
- Run baseline verification without corral models

20th November (Milestone 2):

- Experiment and prepare Corral models.
- Run two subsequent verifications
    - With Corral with manually written models/stubs for library calls
    - With Corral using AV

References

[1] Haas, Andreas & Rossberg, Andreas & Schuff, Derek & Titzer, Ben & Holman, Michael & Gohman, Dan & Wagner, Luke & Zakai, Alon & Bastien, Jf. (2017). Bringing the web up to speed with WebAssembly. 185-200. 10.1145/3062341.3062363.
[2] Zakai, Alon. (2011). Emscripten: an LLVM-to-JavaScript compiler. 301-312. 10.1145/2048147.2048224.
[3] Das, Ankush & Lahiri, Shuvendu & Lal, Akash & Li, Yi. (2015). Angelic Verification: Precise Verification Modulo Unknowns. 10.1007/978-3-319-21690-4_19.
[4] Rakamaric, Zvonimir & Emmi, Michael. (2014). SMACK: Decoupling Source Language Details from Verifier Implementations. 8559. 106-113. 10.1007/978-3-319-08867-9_7.
[5] Haran, Arvind & Carter, Montgomery & Emmi, Michael & Lal, Akash & Qadeer, Shaz & Rakamaric, Zvonimir. (2015). SMACK+Corral: A Modular Verifier. 451-454. 10.1007/978-3-662-46681-0_42.
[6] Leino, K Rustan M & Rümmer, Philipp. (2010). A Polymorphic Intermediate Verification Language: Design and Logical Encoding. 312-327. 10.1007/978-3-642-12002-2_26.