Abstract

End-to-End Translation Validation is the problem of verifying the executable code generated by a compiler against the corresponding input source code for a single compilation. This becomes particularly hard in the presence of dynamically-allocated local memory where addresses of local memory may be observed by the program. In the context of validating the translation of a C procedure to executable code, a validator needs to tackle constant-length local arrays, address-taken local variables, address-taken formal parameters, variable-length local arrays, procedure-call arguments (including variadic arguments), and the `alloca()` operator.

We make the following contributions in our work:

1. A formalization of the execution semantics for an unoptimized intermediate representation (IR) of a C program and its compiled 32-bit x86 assembly in the presence of dynamically (de)allocated local memory. This includes modeling of the various dynamic allocation constructs in C, such as address-taken local variables, constant- and variable-length local arrays, address-taken formal parameters, procedure-call arguments (including variadic arguments), and the GCC `alloca()` operator.

2. A notion of correct translation from the IR to the assembly through a refinement definition. The definition incorporates the concept of undefined behavior (UB) within the IR program, originally translated from C, where refinement is permitted to hold

trivially.

3. An algorithm that converts the correct translation check to first-order logic queries over bitvectors, arrays, and uninterpreted functions that can be discharged using off-the-shelf SMT solvers. The algorithm is capable of operating in both blackbox and whitebox modes, with the blackbox mode enabling its usage with third-party compilers that may not employ a specific allocation strategy, such as preallocation. In particular, we are perhaps the first to enable support for dynamic stack allocation strategy for procedure-call arguments used by almost all production compilers (e.g., GCC, Clang/LLVM).

4. A prototype implementation of the algorithm and its comprehensive evaluation on a set of diverse benchmarks, including both micro-benchmarks and a real-world `bzip2` program. Our prototype performs blackbox translation validation of C procedures with up to 100+ SLOC against their corresponding assembly implementations with up to 140+ instructions generated by an optimizing production compilers (such as GCC, Clang/LLVM, ICC) with complex loop and vectorizing transformations.