
CwicSpec

Aditya Thalang
Independent

With
Apart Research

Abstract

We built a prototype that conjectures candidate specifications for C functions by generating inputs, observing outputs with KLEE, and searching for equivalences between expressions. Inspired by QuickSpec, the project explores whether theory exploration can be applied to C programs. Our implementation is limited by insufficient search-space pruning.

1. Introduction

The adoption of formal verification methods has been limited by the high effort required to annotate code with meaningful specifications. Automated conjecturing and theory exploration offer a promising way to reduce this burden by inferring candidate specifications from observed program behavior. Tools such as QuickSpec have shown that this approach can be effective, rapidly generating useful algebraic laws for functional programs. However, as far as we are aware, no comparable tool exists for C. This leaves a gap for legacy systems code, where verification is especially valuable but specifications are often difficult and costly to write by hand.

Our main contribution is a system for automated conjecturing over C programs. Inspired by QuickSpec, the system generates candidate specifications for C functions from observed behavior

2. Related Work

Theory exploration: QuickSpec uses testing to observe equations between Haskell functions. Related tools such as Speculate extend this to conditional equations and inequalities, while HipSpec uses QuickSpec-generated conjectures as lemmas for inductive proof. Our work follows this theory-exploration approach, but targets C and effectful functions rather than pure functions.

Specification inference: There is also work on inferring specifications using program logics. Bi-abductive separation-logic analysis infers preconditions and frames for heap-manipulating programs, and Quiver infers functional-correctness specifications in separation logic. KindSpec uses symbolic execution in the K framework to infer specifications for programs in KernelC, a subset of C. Our work conjectures specifications for C code without explicit semantics for effects.

Trace mining: A separate line of work recovers temporal specifications from program traces. Perracotta mines temporal API rules from imperfect execution traces, while Texada mines general Linear Temporal Logic properties from logs. Our work is limited to equivalences between C function outputs.

3. Methods

We implemented a system that takes C functions, generates test inputs, observes outputs, and searches for candidate equivalences between generated terms. The design takes inspiration from QuickSpec’s execution-based conjecturing loop, using KLEE for observations.

4. Results

We ran the prototype on examples such as a deficient “ring”. We found it can produce candidate equivalences but the search algorithm is prohibitively slow.

5. Discussion and Limitations

Our implementation does not model effects which restricts the approach to operating best on deterministic functions. Input generation is also limited so it may infer spurious laws when the generated test cases are not diverse enough.

The search space is not sufficiently constrained, making the prototype slow to apply unlike existing work. We also did not perform a systematic evaluation against existing specification-inference tools.

Code and Data

- **Code repository:** <https://github.com/athalang/spsh/releases/tag/submission>

LLM Usage Statement

We used Claude and Codex to assist with writing the codebase and generated code was manually reviewed before inclusion. We also used ChatGPT 5.5 to copy-edit our report.