Project Proposal: Universal Program Input Normalization

July 15, 2025

Project Motivation

Program analysis often becomes complex due to the diverse nature of program input interfaces. Recent optimization-based techniques for program analysis propose transforming each program P into a common intermediate representation R, such that minimizing R solves the analysis problem for P. However, designing such optimization frameworks is inherently challenging when the input space of the program varies significantly—ranging from integers, strings, pointers, complex data structures, or hybrids thereof.

Imagine if all programs accepted inputs from a uniform domain. This homogeneity would drastically simplify the analysis pipeline.

This project aims to address this challenge by developing a program input normalizer that allows all programs to expose the same input interface. This approach will enable standardized optimization-based analysis techniques to be applied uniformly across a wide class of programs.

Problem Statement

Given a program with an input space A and output space B, the goal is to construct a decoder function:

$$\text{decoder} : \text{Bytes} \to A$$

so that the original program can be composed with this decoder to yield:

$$\text{composed\_program} : \text{Bytes} \to B$$

This transformation ensures that any program can be treated as a function from a standard input space (i.e., `Bytes`) to its native output space.

Illustrative Example

Consider a C function implementing a distance calculation:

```
double dist(point* a, point* b);
```

Here, the input consists of two pointers to `point` structures. The objective is to define a function:

$$\text{decoder} : \text{Bytes} \to (\texttt{point*}, \texttt{point*})$$

and hence transform `dist` into a function of type:

$$\text{composed\_dist} : \text{Bytes} \to \texttt{double}$$

The implementation details—especially regarding the encoding and decoding of pointer-based structures in C—are part of the design decisions to be explored.

Tools and Technologies

- Google Protocol Buffers (Protobuf) for defining and serializing structured data.

Challenges and Considerations

- Handling various data representations and memory layouts in low-level languages like C.
- Designing robust and extensible decoders that generalize across program types.
- Ensuring that the transformation preserves the semantics of the original program.

Success Criteria

1. A well-documented final report detailing methodology, implementation, and evaluation.
2. A presentation comprising:
    - A minimal working demo.
    - A demo involving a real-world application.
    - Performance benchmarks of the normalized interface approach.
3. A publicly accessible code repository with:
    - Clear README instructions.
    - Well-organized file structure.
    - Steps for reproducing experiments and benchmarks.