Memory Deallocation and Safety in TIP

Shihe Wang, Simeng Hao

May 4, 2023

Abstract

This report presents the implementation of a new *free* statement for the TIP language, which enables deallocation of heap memory in TIP. To address the memory safety issues arising from deallocation, we also introduce an LLVM pass for detecting use-after-free and double frees in programs written in TIP. The pass first conducts a points-to analysis to determine the possible memory allocations corresponding to each relevant variable, and then checks for memory access violations. The report describes the overall approach, implementation details, and provides examples to demonstrate the effectiveness of the proposed solution.

1 Introduction

In TIP's original design, it lacks a proper memory management mechanism, making it difficult to manage dynamic memory allocations and deallocations. To address this issue, we propose a new *free* statement for the TIP language and an LLVM pass for detecting memory issues, such as use-after-free and double frees.

2 Proposed Solution

Our solution consists of the following components:

1. Implementing a new *free* statement for the TIP language.
2. Developing an LLVM pass to perform points-to analysis.
3. Analyzing memory access instructions for potential issues.

2.1 New *Free* Statement for TIP

In this project, we have extended the TIP language by adding a new *free* statement. This statement allows programmers to deallocate memory that was previously allocated on the heap, giving them more control over memory management. The syntax for the new *free* statement is as follows:

```
free(<variable>);
```

Listing 1: Syntax for the new *free* statement

2.2 LLVM Pass for Points-to Analysis

The LLVM pass we developed performs a points-to analysis on the TIP program to identify which variables may point to stack-allocated or heap-allocated cells. The analysis uses points-to constraints and a worklist algorithm to compute the possible set of memory locations each pointer variable can point to. The result of the analysis is a mapping from pointer variables to sets of memory locations they may point to.

2.3 Memory Access Analysis

The final component of our solution is still under development. It involves examining all memory access instructions in the program to determine if they are legal. In particular, it checks whether memory