Memory unsafety in Stak Scheme virtual machines

@raviqqe

June 7, 2025

Contents

- Stak Scheme
- Progress
  - Library system in `eval` procedure
  - `case-lambda` syntax
- Memory unsafety in virtual machines
- Future work

Stak Scheme

- A bytecode compiler and virtual machine (VM) for Scheme
  - The compiler is written in Scheme.
  - The VM is written in Rust.
- It aims to support the R7RS-small standard.
- Forked from Ribbit Scheme

References

- GitHub
- Website

Progress

- Library system in `eval` procedure
- `case-lambda` syntax

Library system in `eval` procedure

- Stak Scheme now supports the `define-library` syntax in the `eval` procedure.
- The `eval` procedure creates a new library environment for a given library definition.
- It also allows later calls to the `eval` procedure to import the defined libraries.
- It is implemented by sharing logic of the library system in the bytecode compiler.
  - Compiler inception again :)

VM memory unsafety

What is missing in Stak Scheme virtual machine?

- Ribbit Scheme's design achieves simplicity, performance, portability, and extensibility at the same time.
  - A Compact and Extensible Portable Scheme VM
  - Stak Scheme adopts the same design and architecture of the language processor.
- What is missing??
  - Security!

VM memory unsafety in Stak Scheme

- The host language of Rust is memory safe.
- So the virtual machine of Stak Scheme is memory safe.
- However, Scheme programs are not memory safe in terms of memory on the virtual machine.
  - You can violate invariants of the VM memory relatively easily.
  - Today, we call this *VM memory unsafety*.
- It is intentionally VM memory unsafe.
  - Performance gets higher without type checks.
  - The unsafety enables primitive operations in bytecodes.
    - e.g. direct manipulation of stacks

Current status

- For both host and VM memory safety
- Checks in the current implementation
    i. ❌ Language-level type checks
    ii. ❌ Primitive type checks
        - i.e. cons or number
    iii. ✅ Index bound check on memory read/write
    iv. ❌ Index bound check on pointer construction

Upcoming plans

- 1 in Rust?
- 4 instead of 3?

Future work

- Unicode support
- `include` syntax
- Tree shaking
- Synchronous and asynchronous APIs in the same crate

Summary

- Building Scheme is fun! 🥳