

Stak Scheme

@raviqqe

~~August 13, 2023~~ September 10, 2023

Contents

- Overview
- Virtual Machine (VM)
- Bytecodes
- Compiler
- What's implemented so far

Overview

- Stak Scheme
 - There is a typo. (no "c")
 - <https://github.com/raviqqe/stak>
- Scheme runs on a machine stack.
 - No `std` and no `alloc`
- Compiler written in Scheme + VM written in Rust
- Based on [Ribbit Scheme](#)
 - Stak VM does not pursue portability.
 - The VM is specialized for implementation in system programming languages.

Virtual Machine (VM)

- Stack machine
- Von Neumann architecture (?)
 - All bytecodes, stack, heap objects are on VM heap.
- Written in Rust
- No `unsafe` so far

Virtual Machine (VM)

State

- Program counter
 - Points to bytecodes running currently
- Stack
 - Represented as a list
- Symbols
 - Represented as a list of pairs
- Heap (as an unboxed array)
 - If someone wants to run a VM on actual heap, they can simply box it with, for example, `Box::new()` .

Bytecodes

- Mostly borrowed from [Ribbit Scheme](#)
- Represented as lists
- Core instructions
 - `call` : Procedure calls
 - `set` : Set global/local variables
 - `get` : Get global/local variables
 - `constant` : Push constants
 - `if` : Branch based on condition values
- Primitives: `rib` , `cons` , `close` , ...

Compiler

Main routine

```
(write-target (encode (compile (expand (read-source))))))
```

- `read-source` reads S-expressions from stdin.
- `expand` expands syntax sugar (e.g. `let`, `letrec`, etc.)
- `compile` compiles S-expressions into bytecodes.
- `encode` encodes bytecodes on memory into bytes.
- `write-target` writes encoded bytecodes into stdout.

What's implemented so far

Syntax

- Function/variable definitions
- Closures
- Binding expressions
 - e.g. `let`, `let*`, `letrec`, ...
- Conditional expressions
 - e.g. `if`, `cond`, `when`, ...
- `begin` block

What's implemented so far

Built-ins

- Arithmetic operations
- Comparison operations
- Boolean operations
- List operations (`car` , `cdr` , `cons` , `map` , `length` ...)
- Continuation (`call/cc`)
- `error`

What's implemented so far

Types

- Signed 63-bit integer
- Boolean
- Pair / Null
- Symbol
- Character
- String
- Vector / Bytevector (as list)
- Procedure

What's implemented so far from the last missed meetup

- `define-syntax` , `let-syntax` , `letrec-syntax`
- Non-hygienic `syntax-rules`
- Efficient representation of argument counts at call sites

Next tasks...

- Hygienic `syntax-rules`
- Quasi-quotation
- Record type
- `write` and `display`
- `read`
- `cond-expand`

Summary

- Building Scheme is fun.

References

- [Ribbit Scheme](#)
- [R7RS](#)