Bytecode encoding v2 in Stak Scheme

@raviqqe

December 22, 2024

Contents

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

Progress

- Bytecode encoding v2

Bytecode encoding v1 in Stak Scheme

- In Stak (and Ribbit) Scheme, everything is a list.
- Bytecodes and data values are represented by cons's or scalars (numbers.)
  - Only `if` instructions branch into two preceding instruction lists.
  - Immediate values in instructions can be of any data type.
- A program can be considered as a DAG composed of cons's with instruction codes interleaved with data.

We can just encode/decode DAG's of nodes with zero to two edges!

References

- A R4RS Compliant REPL in 7 KB, Léonard et al.

Bytecode encoding v1

- It is roughly borrowed from Ribbit Scheme.

Decoding

1. Expand a symbol table.
    - Symbols may or may not have their string representations.
2. Decode instruction lists recursively as lists into memory.
    - On encoding, when we hit symbols or non-number constants, we look up the symbol table and store their indices into decoded instructions.
3. On initialization, we initialize constants by executing constant initialization logic attached at the beginning of the program.

Bytecode encoding v2

- The new bytecode format is aimed for:
  - Simpler decoding
  - Faster startup time

Decoding

1. Decode instruction lists recursively as lists into memory.
   - Including both instructions and immediate values.
2. Done 😄

References

- https://github.com/raviqqe/til/tree/main/dag-encoder

Bytecode encoding v2

Pros

- The new encoding algorithm:
  - Doesn't have any global symbol table during encoding/decoding.
  - Eliminates constant initialization at runtime.
    - They are natively marshalled and serialized into bytecodes.

Cons

- Slightly bigger bytecode sizes
  - Up to around 1.5 times

Benchmarks

`stak`, the interpreter

```
Benchmark 1: /Users/raviqqe/src/github.com/raviqqe/stak/target/release/stak ~/foo.scm
  Time (mean ± σ):     127.4 ms ±   0.9 ms    [User: 122.9 ms, System: 3.8 ms]
  Range (min … max):   126.6 ms … 130.1 ms    23 runs

Benchmark 2: ~/worktree/7a1181edfad9f3e5/target/release/stak ~/foo.scm
  Time (mean ± σ):     196.3 ms ±   2.9 ms    [User: 190.8 ms, System: 4.4 ms]
  Range (min … max):   189.3 ms … 199.1 ms    15 runs

Relative speed comparison
        1.00          /Users/raviqqe/src/github.com/raviqqe/stak/target/release/stak ~/foo.scm
        1.54 ±  0.03  ~/worktree/7a1181edfad9f3e5/target/release/stak ~/foo.scm
```

Benchmarks

`mstak`, the minimal interpreter

```
Benchmark 1: /Users/raviqqe/src/github.com/raviqqe/stak/cmd/minimal/target/release/mstak ~/foo.scm
  Time (mean ± σ):      72.6 ms ±   1.9 ms    [User: 68.1 ms, System: 3.7 ms]
  Range (min … max):    70.4 ms …  75.9 ms    40 runs

Benchmark 2: ~/worktree/7a1181edfad9f3e5/cmd/minimal/target/release/mstak ~/foo.scm
  Time (mean ± σ):     105.9 ms ±   3.3 ms    [User: 101.2 ms, System: 3.8 ms]
  Range (min … max):   102.1 ms … 111.0 ms    26 runs

Relative speed comparison
        1.00          /Users/raviqqe/src/github.com/raviqqe/stak/cmd/minimal/target/release/mstak ~/foo.scm
        1.46 ±  0.06  ~/worktree/7a1181edfad9f3e5/cmd/minimal/target/release/mstak ~/foo.scm
```

Future work

- ~~Faster startup time~~ Finally!
- Easier integration with Rust
- Better compatibility with the R7RS small

Summary

- Building a bytecode encoder is fun! 😃