`eval`ing macros in Stak Scheme

@raviqqe

June 2, 2024

Contents

- Stak Scheme
- Macros in Scheme
- Implementation in Stak Scheme
- Demo
- 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 R7RS-small.

Macros in Scheme

- Macros are defined in source codes.
  - Usually in `(define-library)` forms.
- Macros in Scheme can be expanded at compile time.
  - Stak's compiler does that.

```
(define-syntax or
  (syntax-rules ()
    ((_)
      #f)

    ((_ test)
      test)

    ((_ test1 test2 ...)
      (let ((x test1))
        (if x x (or test2 ...))))))
```

Macros in `eval`

- The `eval` procedure evaluates S-expressions.
  - The expressions can be primitive values, procedure calls, or macro expansions.
- No macros or their information at runtime when they are expanded at compile time!
- We need pass macros into runtime codes in some way.

Implementation of macros in `eval` in Stak Scheme

`($$macros)` primitive

- Used in Scheme source codes.
- Expanded by a compiler into macro rules.
  - Macro rules are represented by lists, symbols, and literals.
- At runtime, macros are compiled to macro transformers of procedures.

Implementation of macros in `eval` in Stak Scheme

Macro expansion in `eval`

- The `eval` procedure expands all macros in a given expression first.
- Then, it compiles the expression into a temporary procedure and calls it.

Demo

Future work

- Deduplication of codes between a compiler and the `(scheme eval)` library

Summary

- Building macros in `eval` is fun!