Parsing Scheme

(+ (* 2 3) 1)
Compiling Scheme

+ 
  * 
  1 
  2 3

+ 
  constant 
  3

+ 
  refer 
  2

apply 
  + 
  refer 
  1
Compiling Scheme

START

pair? yes -> make-test yes -> make-return

pair? no -> symbol? yes -> make-refer

symbol? no -> make-constant


lambda? no -> make-apply

define? no -> make-constant

make-assign

make-close

make-apply

make-frame

make-argument
```plaintext
bytecode
  frame
    constant
    argument
    constant
    argument
    constant
    argument
  refer
  apply
    * 
    refer
  apply
    +

frame
  halt
  pc
  val
  .
  .
  .
  argument
  val
  .
  .
  .

constant
  3

argument
  2

registers
  pc
  arguments
  environment
  frames

apply

environment
  . .

argument stack
  pc
  arg
  env
  .
  .
  .

frame stack
  pc
  arg
  env
  .
  .
  .
```
Frame Bytecode

- Set program counter to second continuation
- Push registers onto frame stack
- Set program counter to first continuation
Constant Bytecode

- Set accumulator to value of operand
- Set program counter to continuation
Argument Bytecode

- Push accumulator onto argument stack
- Set program counter to continuation
Refer Bytecode

- Lookup value of operand in environment and store it in accumulator
- Set program counter to continuation
Apply Bytecode (primitive)

- Apply primitive to vals on argument stack
- Pop frame stack
Apply Bytecode (closure)

- Make application frame using closure's argument list (vars) and argument stack (vals)
- Push application frame onto environment
- Set program counter to body of closure
Return Bytecode

- Pop frame stack
Test Bytecode

- If accumulator is not \#f then set program counter to first continuation else set program counter to second continuation
Assign Bytecode

- Make definition using operand and accumulator
- Add definition to top frame of environment
- Set program counter to continuation
Close Bytecode

- Make closure using operands and environment
- Set program counter to continuation
Halt Bytecode

- halt
Parsing Scheme

(+ (* 2 3) 1)

Diagram of expression parsing.
Compiling Scheme
Compiling Scheme

START

- pair? yes → make-test
- pair? no → symbol?
  - yes → if?
  - no → make-apply
- symbol? yes → make-refer
- symbol? no → make-constant

- make-test yes → make-return
- make-test no → make-close
- make-close yes → make-assign
- make-assign yes → make-constant
- make-constant yes → make-frame
- make-frame
- make-frame 0 → make-argument
- make-frame 1 → make-argument
- make-frame 2 → make-argument

- if? yes → lambda?
- if? no → define?
- lambda? yes → quote?
- lambda? no → make-apply
- define? yes → make-constant
- define? no → make-constant
- make-constant

- make-constant

Interpreting Bytecode

- frame
  - constant
    - argument
      - constant
        - argument
          - apply
            - apply
              +
              +

- frame
  - constant
    - argument
      - constant
        - argument
          - apply
            - apply
              +
              +

pc = halt
acc = 7
Frame Bytecode

- Set program counter to second continuation
- Push registers onto frame stack
- Set program counter to first continuation
Constant Bytecode

- Set accumulator to value of operand
- Set program counter to continuation
**Argument Bytecode**

- Push accumulator onto argument stack
- Set program counter to continuation
Refer Bytecode

- Lookup value of operand in environment and store it in accumulator
- Set program counter to continuation
Apply Bytecode (primitive)

- Apply primitive to vals on argument stack
- Pop frame stack
Apply Bytecode (closure)

• Make application frame using closure's argument list (vars) and argument stack (vals)
• Push application frame onto environment
• Set program counter to body of closure
Return Bytecode

- Pop frame stack
Test Bytecode

- If accumulator is not `#f` then set program counter to first continuation else set program counter to second continuation
Assign Bytecode

- Make definition using operand and accumulator
- Add definition to top frame of environment
- Set program counter to continuation
Close Bytecode

- Make closure using operands and environment
- Set program counter to continuation
Halt Bytecode

- halt