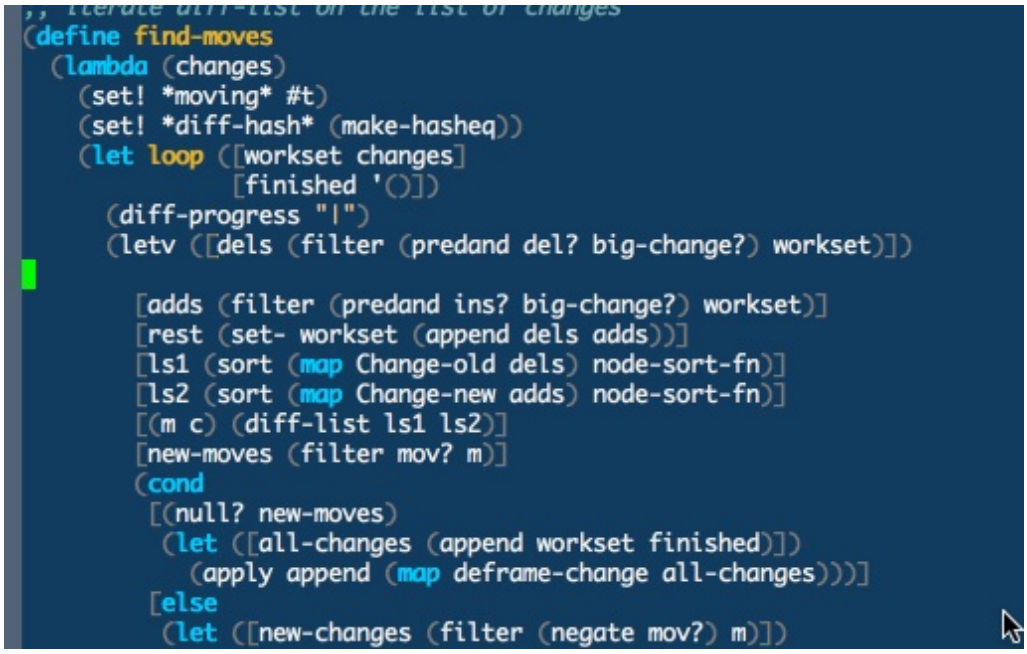
Scheme



Scheme Scheme Scheme

Scheme [Quack]

Scheme

Chez Scheme

Scheme R. Kent Dybvig Chez Scheme Scheme Chez Scheme Chez Scheme

https://github.com/cisco/ChezScheme

Linux Mac

```
./configure
make
sudo make install
```

30

Racket

Racket

http://racket-lang.org

Ubuntu Racket

Paredit mode

Scheme Emacs Paredit mode “” Scheme Lisp

Paredit mode

http://mumble.net/~campbell/emacs/paredit.el

~/.emacs.d ~/.emacs

```
(add-to-list 'load-path "~/.emacs.d")
```

```
(autoload 'paredit-mode "paredit"
  "Minor mode for pseudo-structurally editing Lisp code."
  t)
```

M-x paredit-mode C-h m “”

scheme mode

Scheme cmuscheme Scheme .emacs

```
;;;;;;;;;;;;
;; Scheme
;;;;;;;;;;;;

(require 'cmuscheme)

;; push scheme interpreter path to exec-path
(push "/Applications/Racket/bin" exec-path)

;; scheme interpreter name
(setq scheme-program-name "racket")

;; bypass the interactive question and start the default interpreter
(defun scheme-proc ()
  "Return the current Scheme process, starting one if necessary."
  (unless (and scheme-buffer
               (get-buffer scheme-buffer)
               (comint-check-proc scheme-buffer))
    (save-window-excursion
      (run-scheme scheme-program-name)))
  (or (scheme-get-process)
      (error "No current process. See variable `scheme-buffer'")))

(defun switch-other-window-to-buffer (name)
    (other-window 1)
    (switch-to-buffer name)
    (other-window 1))

(defun scheme-split-window ()
  (cond
   ((= 1 (count-windows))
    (split-window-vertically (floor (* 0.68 (window-height))))
    ;; (split-window-horizontally (floor (* 0.5 (window-width))))
    (switch-other-window-to-buffer "*scheme*"))
   ((not (member "*scheme*"
               (mapcar (lambda (w) (buffer-name (window-buffer w)))
                       (window-list))))
    (switch-other-window-to-buffer "*scheme*"))))

(defun scheme-send-last-sexp-split-window ()
  (interactive)
  (scheme-split-window)
  (scheme-send-last-sexp))

(defun scheme-send-definition-split-window ()
  (interactive)
  (scheme-split-window)
  (scheme-send-definition))

(add-hook 'scheme-mode-hook
  (lambda ()
    (paredit-mode 1)
    (define-key scheme-mode-map (kbd "<f5>") 'scheme-send-last-sexp-split-window)
    (define-key scheme-mode-map (kbd "<f6>") 'scheme-send-definition-split-window)))
```

Scheme Paredit mode F5 “S”S F5

Paredit mode

Paredit mode

Paredit mode

1. `C-right`: Ctrl ""S

   `(a b c) (d e)` `(a b c)` `C-right` `(a b c (d e))` `(d e)` "" `(a b c)`

2. `M-r`:

   let

   ```
   (let ([x 10])
    (* x 2))

   (* x 2) M-r

   (* x 2)

   (let ([x 10]) ...) ""
   ```

Lisp

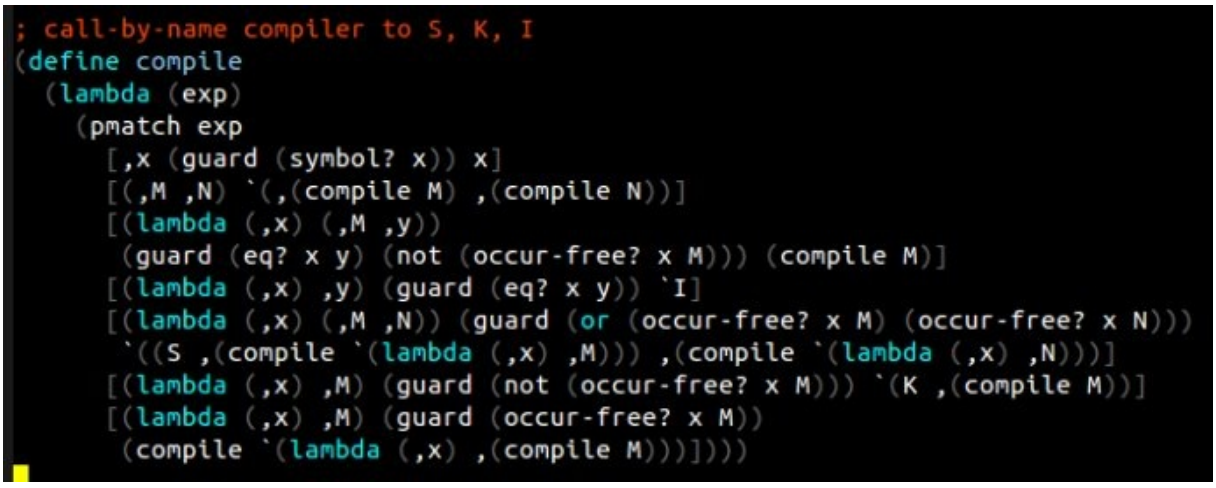
el .emacs.d:

https://www.dropbox.com/s/v0ejctd1agrt95x/parenface.el

.emacs

```
(require 'parenface)
(set-face-foreground 'paren-face "DimGray")
```

Scheme



```
; call-by-name compiler to S, K, I
(define compile
  (lambda (exp)
    (pmatch exp
      [,x (guard (symbol? x)) x]
      [(,M ,N) `(,(compile M) ,(compile N))]
      [(lambda (,x) (,M ,y))
       (guard (eq? x y) (not (occur-free? x M))) (compile M)]
      [(lambda (,x) ,y) (guard (eq? x y)) `I]
      [(lambda (,x) (,M ,N)) (guard (or (occur-free? x M) (occur-free? x N)))
       `((S ,(compile `(lambda (,x) ,M))) ,(compile `(lambda (,x) ,N)))]
      [(lambda (,x) ,M) (guard (not (occur-free? x M))) `(K ,(compile M))]
      [(lambda (,x) ,M) (guard (occur-free? x M))
       (compile `(lambda (,x) ,(compile M)))]))))
```

Scheme