The Stepper

Version 9.4.0.2

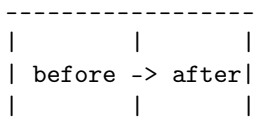
October 20, 2026

1 What is the Stepper?

DrRacket includes an "algebraic stepper," a tool which proceeds through the evaluation of a set of definitions and expressions, one step at a time. This evaluation shows the user how DrRacket evaluates expressions and definitions, and can help in debugging programs. Currently, the Stepper is available in the "Beginning Student" and "Intermediate Student" language levels.

2 How do I use the Stepper?

The Stepper operates on the contents of the frontmost DrRacket window. A click on the "Step" button brings up the stepper window. The stepper window has two panes, arranged as follows:



The first, "before," box, shows the current expression. The region highlighted in green is known as the "redex". You may pronounce this word in any way you want. It is short for "reducible expression," and it is the expression which is the next to be simplified.

The second, "after," box shows the result of the reduction. The region highlighted in purple is the new expression which is substituted for the green one as a result of the reduction. For most reductions, the only difference between the left- and right-hand sides should be the contents of the green and purple boxes.

Please note that the stepper only steps through the expressions in the definitions window, and does not allow the user to enter additional expressions. So, for instance, a definitions buffer which contains only procedure definitions will not result in any reductions.

3 How Does the Stepper work?

In order to discover all of the steps that occur during the evaluation of your code, the Stepper rewrites (or "instruments") your code. The inserted code uses a mechanism called "continuation marks" to store information about the program's execution as it is running, and makes calls to the Stepper before, after and during the evaluation of each expression, indicating the current shape of the program.

What does this instrumented code look like? For the curious, here's the expanded version of `(define (f x) (+ 3 x))` in the beginner language [*]:

```
(module #%htdp (lib "lang/htdp-beginner.rkt")
  (#%plain-module-begin
   (define-syntaxes (f)
     (#%app make-first-order-function
            'procedure
            '1
            (quote-syntax f)
            (quote-syntax #%app)))
   (define-values (test~object) (#%app namespace-variable-
value 'test~object))
   (begin
     (define-values (f)
       (with-continuation-mark "#<debug-key-struct>"
         (#%plain-lambda () (#%plain-app "#<procedure:...rivate/marks.rkt:70:2>"))
         (#%plain-app
          call-with-values
          (#%plain-lambda ()
            (with-continuation-mark "#<debug-key-struct>"
              (#%plain-lambda () (#%plain-app
                                  "#<procedure:...rivate/marks.rkt:70:2>"
                                  (#%plain-lambda () beginner:+)))
              (#%plain-app
               "#<procedure:closure-storing-proc>"
               (#%plain-lambda (x)
                 (begin
                   (let-values ([(arg0-1643 arg1-1644 arg2-1645)
                                 (#%plain-app
                                  values
                                  "#<*unevaluated-struct*>"
                                  "#<*unevaluated-struct*>"
                                  "#<*unevaluated-struct*>")])
                     (with-continuation-mark "#<debug-key-struct>"
                       (#%plain-lambda ()
                         (#%plain-app
                          "#<procedure:...rivate/marks.rkt:70:2>"
```

```
                          (#%plain-lambda () beginner:+)
                          (#%plain-lambda () x)
                          (#%plain-lambda () arg0-1643)
                          (#%plain-lambda () arg1-1644)
                          (#%plain-lambda () arg2-1645)))
                        (begin
                          (#%plain-app "#<procedure:result-exp-
break>")
                          (begin
                            (set! arg0-1643
                                  (with-continuation-mark "#<debug-
key-struct>"
                                    (#%plain-lambda ()
                                      (#%plain-app
                                       "#<procedure:...rivate/marks.rkt:70:2>"))
                                    beginner:+))
                            (set! arg1-1644
                                  (with-continuation-mark "#<debug-
key-struct>"
                                    (#%plain-lambda ()
                                      (#%plain-app
                                       "#<procedure:...rivate/marks.rkt:70:2>"))
                                    '3))
                            (set! arg2-1645
                                  (with-continuation-mark "#<debug-
key-struct>"
                                    (#%plain-lambda ()
                                      (#%plain-app
                                       "#<procedure:...rivate/marks.rkt:70:2>")) x))
                            (begin
                              (#%plain-app "#<procedure:normal-
break>")
                              (with-continuation-mark "#<debug-key-
struct>"
                                (#%plain-lambda ()
                                  (#%plain-app
                                   "#<procedure:...rivate/marks.rkt:70:2>"
                                   (#%plain-lambda () arg0-1643)
                                   (#%plain-lambda () arg1-1644)
                                   (#%plain-lambda () arg2-1645)))
                                (if (#%plain-app
                                     "#<procedure:annotated-proc?>"
                                     arg0-1643)
                                    (#%plain-app
                                     arg0-1643
                                     arg1-1644
```

```
                                      arg2-1645)
                                     (#%plain-app
                                      call-with-values
                                      (#%plain-lambda ()
                                        (#%plain-app arg0-1643 arg1-
1644 arg2-1645))
                                      (#%plain-lambda args
                                        (#%plain-app
                                         "#<procedure:result-value-
break>"
                                         args)
                                        (#%plain-app
                                         "#<procedure:apply>"
                                         values
                                         args))))))))))))
                 (#%plain-lambda ()
                   (#%plain-app
                    "#<procedure:...rivate/marks.rkt:70:2>"
                    (#%plain-lambda () beginner:+))) #f)))
            (#%plain-lambda args
              (#%plain-app "#<procedure:apply>" values args)))))
       (#%plain-app "#<procedure:exp-finished-break>"
                    (#%plain-app
                     list
                     (#%plain-app
                      list
                      "#<procedure:...ate/annotate.rkt:1256:93>"
                      #f
                      (#%plain-lambda () (#%plain-app list f))))))))

(let-values ([(done-already?) '#f])
  (#%app dynamic-wind void
         (lambda () (#%app dynamic-require ''#%htdp   '#f))
         (lambda () (if done-already?
                        (#%app void)
                        (let-values ()
                          (set! done-already? '#t)
                          (#%app test*)
                          (#%app current-namespace
                                 (#%app module->namespace
                                        ''#%htdp)))))))
```

[*] : In order to allow things like

```
#<procedure:apply>
```

in scribble, I've taken the cheap solution of wrapping them in quotes. These are not actually strings, they're opaque 3D syntax elements.