

CS 135 – L16: Local and Application

Luke Lu • 2025-11-11

Local Definition



```
(local
  [(define ...)]
  [(define ...)])
  ...
  (local scope code)
)
```

Example:

Heron's Formula

```
define (heron a b c)
  (local [(define s (/ (+ a b c) 2))]
    (sqrt (* s (- s a) (- s b) (- s c)))))
```



Warning – Functions under local cannot be checked with check-expect

α Conversion

1. Choose a fresh name for each definition in the local.
2. Substitute that fresh name everywhere it is used in the local.
3. Lift the definition to the top level.
4. Replace (local [...] expression) with just the expression.

Stepper example:

The screenshot shows the DrRacket Stepper window. At the top, there are navigation buttons: Beginning, Previous Call, Previous, Selected, Next, Next Call, and End. On the right, it shows "1/10" and the DrRacket logo. The main area displays two blocks of code side-by-side. The left block is highlighted in green and contains the original code:`(local
 ((define x 6)
 (define y 2)
 (define z 1)
 (define (f x) (+ x 1))
 (define (g x y) (+ x y)))
 (+ (f x) (g y z)))`

```
The right block is highlighted in purple and shows the result after alpha conversion:
```

`(define x_0 6)
(define y_0 2)
(define z_0 1)
(define (f_0 x) (+ x 1))
(define (g_0 x y) (+ x y))
(+ (f_0 x_0) (g_0 y_0 z_0))`

Benefits of Local

1. Clarity: Naming subexpressions
2. Encapsulation: Hiding stuff
3. Scope: Reusing names
4. Efficiency: Avoiding recomputation

Encapsulation

Encapsulation describes the general programming principle that we should “encapsulate” (or hide away) implementation details that are not relevant to how a function is used (“information hiding”).

Local bindings are not visible, and have no effect outside the local expression. Thus, they can “hide” information from other parts of a program.

Example:

```
(define (rev lst)
  (local
    [(define (rev/accumulate lst accumulator)
       (cond [(empty? lst) accumulator]
             [else (rev/accumulate (rest lst) (cons (first lst) accumulator))])])
     (rev/accumulate lst empty)))

(check-expect (rev '(d c b a)) '(a b c d))
```

Efficiency

Serve as memoization/DP/caching

Binary Tree Example

```
(define (bt-path label bt)
  (cond [(empty? bt) empty]
        [= (get-label bt) label] (list 'found))
        [else
         (local
           [(define lpath (bt-path label (get-left bt)))]
           (cond [(empty? lpath)
                  (local
                    [(define rpath (bt-path label (get-right bt)))]
                    (cond [(empty? rpath) empty]
                          [else (cons 'right rpath)])]
                  [else (cons 'left lpath)]))))]
```

Stepper Rule

Examples:/

```
(local [(define x 10) (define y (+ x 5))] (* x y))

(define x_0 10)
(define y_0 (+ x_0 5))
(* x_0 y_0)
```

```
(define x_0 10)
(define y_0 (+ 10 5))
(* x_0 y_0)
```

```
(define x_0 10)
(define y_0 15)
(* 10 y)
(* 10 15)
```

150

```
(local [(define z 3)] (+z (local [define z 7] (* z 2))))
```

```
(define z_0 3)
(+ z_0 (define z_1 7) (* z_1 2)))
```

```
(define z_0 3)
(define z_1 7)
(+ 3 (* z_1 2)))
```

```
(define z_0 3)
(define z_1 7)
(+ 3 (* 7 2))
```

```
(+ 3 14)
```

17