# ERRATA
## COMPILING LAMBDA CALCULUS

**Page 15**

The table on page 15 should mention $\underline{\lambda x \lambda y M}$ as a shorthand for $\lambda x.\, \lambda y M$.

**Page 18**

Substitution rule (S6) contains ...

should be

Substitution rule $\underline{(\mathrm{S5})}$ contains ...

**Page 31**

$\rightarrow\ \lambda fx.(\lambda x_1.\, \lambda h.\, h\ \mathbf{(f\ (f(x_1 f)))})\ (\lambda ux)\ (\lambda uu)$

should be

$\underline{\twoheadrightarrow}\ \lambda fx.(\lambda x_1.\, \lambda h.\, h\ \mathbf{(f\ (f(x_1 f)))})\ (\lambda ux)\ (\lambda uu)$

**Page 49**

In the definition of (Env1),

$\rightarrow\ (\lambda x_2.[x_1 = N_1]M)N_2 \cdots N_n$

should be

$\rightarrow\ (\lambda x_2 \underline{\cdots x_n}.[x_1 = N_1]M)N_2 \cdots N_n$

**Page 115**

```
            if (P-m > 0) {
                    kk = vector(S)[P-m-2];
```

should be

```
            if (P-m > 0) {
                    restore_vars(vector(S)[P-3]);
                    kk = vector(S)[P-m-2];
```

**Page 126**

```
(lift-prim x))
```

should be

```
(lift-prim `(,(car x) ,@(map exp (cdr x)))))
```

**Page 135**

Variable *tco* is renamed *n* and

```
        if (tco && P > 4) {
                vector(S)[P-5] = vector(S)[P-1];
```

should be

```
        if (n && P > 4) {
                vector(E)[n-1] = vector(S)[P-2];
                vector(S)[P-5] = vector(S)[P-1];
```

**Page 155 (Appendix)**

```
          (let ((m (stack-ref (+ Frame-skip n))))
            (cond ((= n m)
```

should be

```
          (let ((m (stack-ref (+ Frame-skip n))))
            (restore-vars (stack-ref 3))
            (cond ((= n m)
```

**NOTE**

*(The following issue does not affect any examples in the book!)*

The $Scheme_3$ and $LC_n$ compiler in the book fail to compile the following program properly. It should reduce to (*not broken*), but does reduce to (*broken broken*).

$$
\begin{array}{l}
((\lambda q((\lambda h((\lambda f(f(id(\underline{not}\ \underline{broken}))))\\
\qquad((\lambda r((\lambda t(\underline{set}\ r\ t))\\
\qquad\qquad(\lambda x[\mathbf{if}(\underline{pairp}\ x)(\underline{seq}\ (r(\underline{cdr}\ x))(h(\underline{car}\ x)))x]))))\\
\qquad\underline{nil})))\\
\quad(\lambda x(\underline{set}\ q\ (\underline{cons}\ x\ q)))))\\
\underline{nil})
\end{array}
$$

Explanation:

The function $r$ exits using a tail application of $h$, but the runtime code of the compiler restores bindings of outer contexts upon function return, which is never reached in this case. Therefore, the binding $x = (\underline{not}\ \underline{broken})$ of the context of $r$ is never restored after the recursive application of $r$, and hence the $\underline{car}$ of $x = (\underline{broken})$ is incorrectly passed to $h$. This issue affects only recursive functions using the **rec** schema.

Fixes to the $Scheme_3$ compiler are included in the errata above. Fixed versions of the $LC_n$ compilers can be found on the book home page: **http://t3x.org/clc/**.

Here is a patch for the $LC_3$ compiler. It should work for the other $LC_n$ compilers as well.

```
--- lcomp/lc.c.OLD
+++ lcomp/lc.c
@@ -144,12 +144,13 @@
       push(new_atom(T_FUNCTION, fn));
 }

-int apply(int k, int tco) {
+int apply(int k, int n) {
       T = vector(S)[P-1];
       P--;
       if (!function_p(T))
              err("application of non-function");
-      if (tco && P > 4) {
+      if (n && P > 4) {
+             vector(E)[n-1] = vector(S)[P-2];
              vector(S)[P-5] = vector(S)[P-1];
              P-=2;
       }
--- lcomp/lc3.scm.OLD
+++ lcomp/lc3.scm
@@ -134,32 +134,33 @@
   (car (reverse x)))

 (define (tconv x)
-  (define (tc x t)
+  (define (tc x n t)
     (cond ((not (pair? x))
            x)
          ((memq (car x) '(id %ref))
           x)
```

```
           ((eq? (car x) 'lam)
             `(lam ,(cadr x)
-                  ,(tc (caddr x) #t)))
+                  ,(tc (caddr x) (+ 1 n) #t)))
           ((eq? (car x) 'set)
             `(set ,(cadr x)
-                  ,(tc (caddr x) #f)))
+                  ,(tc (caddr x) n #f)))
           ((eq? (car x) 'seq)
             `(seq ,@(map (lambda (x)
-                           (tc x #f))
+                           (tc x n #f))
                         (but-last (cdr x)))
-                  ,(tc (last x) t)))
+                  ,(tc (last x) n t)))
           ((eq? (car x) 'if)
-            `(if ,(tc (cadr x) #f)
-                 ,(tc (caddr x) t)
-                 ,(tc (cadddr x) t)))
+            `(if ,(tc (cadr x) n #f)
+                 ,(tc (caddr x) n t)
+                 ,(tc (cadddr x) n t)))
           (else
             `(,(if t '%tail-apply '%apply)
+              ,n
               ,@(map (lambda (x)
-                       (tc x #f))
+                       (tc x n #f))
                      x)))))
-  (tc x #t))
+  (tc x 0 #t))

 (define (gen x)
   (define n 1)
@@ -236,25 +237,25 @@

   (define (gapp x)
     (let ((a (addr)))
+      (g (cadddr x))
       (g (caddr x))
-      (g (cadr x))
       (emit "K = apply("
             a
             ", "
-            (if (eq? '%apply (car x)) 0 1)
+            (if (eq? '%apply (car x)) 0 (cadr x))
             "); break;")
       (glab a)))
```

```
   (define (gprim1 x)
-    (g (caddr x))
-    (emit* "T = P_" (cadr x) "(vector(S)[P-1]); ")
+    (g (cadddr x))
+    (emit* "T = P_" (caddr x) "(vector(S)[P-1]); ")
     (emit "vector(S)[P-1] = T;"))

   (define (gprim2 x)
-    (g (caddr x))
     (g (cadddr x))
+    (g (car (cddddr x)))
     (emit* "T = P_"
-           (cadr x)
+           (caddr x)
            "(vector(S)[P-2], "
            "vector(S)[P-1]); ")
     (emit* "vector(S)[P-2] = T; ")
@@ -293,9 +294,9 @@
             (gseq x))
           ((eq? (car x) 'if)
            (gif x))
-          ((memq (cadr x) prim1)
+          ((memq (caddr x) prim1)
            (gprim1 x))
-          ((memq (cadr x) prim2)
+          ((memq (caddr x) prim2)
           (gprim2 x))
          (else
           (gapp x))))
```