five.scm

```
5
```

five.cmp

```
((continue)
 (val)
 (val)
 (
  (assign val (const 5))
  (goto (reg continue))
 )
)
```

define.scm

```
(define a 3)
```

define.cmp

```
((env continue)
 (val)
 (val)
 (
  (assign val (const 3))
  (perform (op define-variable!) (const a) (reg val) (reg env))
  (assign val (const a))
  (goto (reg continue))
 )
)
```

define2.scm

```
(define a 3)
(define b a)
```

define2.cmp

```
((env continue)
 (val)
 (val)
 (
  (assign val (const 3))
  (perform (op define-variable!) (const a) (reg val) (reg env))
  (assign val (const a))
  (assign val (op lookup-variable-value) (const a) (reg env))
  (perform (op define-variable!) (const b) (reg val) (reg env))
  (assign val (const b))
  (goto (reg continue))
 )
)
```

if.scm

```
(if 2 3 4)
```

if.cmp

```
((continue)
 (val)
 (val)
 (
  (assign val (const 2))
  (test (op false?) (reg val))
  (branch (label false-branch2))
 true-branch1
  (assign val (const 3))
  (goto (reg continue))
 false-branch2
  (assign val (const 4))
  (goto (reg continue))
 after-if3
 )
)
```

lambda.scm

```
(lambda (x y) (+ x y))
```

lambda.cmp

```
((env continue)
 (val)
 (val)
 (
  (assign val (op make-compiled-procedure) (label entry1) (reg env))
  (goto (reg continue))
 entry1
  (assign env (op compiled-procedure-env) (reg proc))
  (assign env (op extend-environment) (const (x y)) (reg argl) (reg env))
  (assign proc (op lookup-variable-value) (const +) (reg env))
  (assign val (op lookup-variable-value) (const y) (reg env))
  (assign argl (op list) (reg val))
  (assign val (op lookup-variable-value) (const x) (reg env))
  (assign argl (op cons) (reg val) (reg argl))
  (test (op primitive-procedure?) (reg proc))
  (branch (label primitive-branch3))
 compiled-branch4
  (assign val (op compiled-procedure-entry) (reg proc))
  (goto (reg val))
 primitive-branch3
  (assign val (op apply-primitive-procedure) (reg proc) (reg argl))
  (goto (reg continue))
 after-call5
 after-lambda2
 )
)
```

plus.scm

```
(+ 7 12)
```

plus.cmp

```
((env continue)
 (env proc argl continue temp1 temp2 val)
 (proc argl val)
 (
  (assign proc (op lookup-variable-value) (const +) (reg env))
  (assign val (const 12))
  (assign argl (op list) (reg val))
  (assign val (const 7))
  (assign argl (op cons) (reg val) (reg argl))
  (test (op primitive-procedure?) (reg proc))
  (branch (label primitive-branch1))
 compiled-branch2
  (assign val (op compiled-procedure-entry) (reg proc))
  (goto (reg val))
 primitive-branch1
  (assign val (op apply-primitive-procedure) (reg proc) (reg argl))
  (goto (reg continue))
 after-call3
 )
)
```

define3.scm

```
(define a 3)
(define b (+ a 7))
```

define3.cmp

```
((env continue)
 (proc argl temp1 temp2 val)
 (proc argl val)
 (
  (assign val (const 3))
  (perform (op define-variable!) (const a) (reg val) (reg env))
  (assign val (const a))
  (save continue)
  (save env)
  (assign proc (op lookup-variable-value) (const +) (reg env))
  (assign val (const 7))
  (assign argl (op list) (reg val))
  (assign val (op lookup-variable-value) (const a) (reg env))
  (assign argl (op cons) (reg val) (reg argl))
  (test (op primitive-procedure?) (reg proc))
  (branch (label primitive-branch1))
 compiled-branch2
  (assign continue (label after-call3))
  (assign val (op compiled-procedure-entry) (reg proc))
  (goto (reg val))
 primitive-branch1
  (assign val (op apply-primitive-procedure) (reg proc) (reg argl))
 after-call3
  (restore env)
  (perform (op define-variable!) (const b) (reg val) (reg env))
  (assign val (const b))
  (restore continue)
  (goto (reg continue))
 )
)
```

define4.scm

```
(define f (lambda (x y) (+ x y)))
```

define4.cmp

```
((env continue)
 (val)
 (val)
 (
  (assign val (op make-compiled-procedure) (label entry1) (reg env))
  (goto (label after-lambda2))
 entry1
  (assign env (op compiled-procedure-env) (reg proc))
  (assign env (op extend-environment) (const (x y)) (reg argl) (reg env))
  (assign proc (op lookup-variable-value) (const +) (reg env))
  (assign val (op lookup-variable-value) (const y) (reg env))
  (assign argl (op list) (reg val))
  (assign val (op lookup-variable-value) (const x) (reg env))
  (assign argl (op cons) (reg val) (reg argl))
  (test (op primitive-procedure?) (reg proc))
  (branch (label primitive-branch3))
 compiled-branch4
  (assign val (op compiled-procedure-entry) (reg proc))
  (goto (reg val))
 primitive-branch3
  (assign val (op apply-primitive-procedure) (reg proc) (reg argl))
  (goto (reg continue))
 after-call5
 after-lambda2
  (perform (op define-variable!) (const f) (reg val) (reg env))
  (assign val (const f))
  (goto (reg continue))
 )
)
```

application.scm

```
(define f (lambda (x y) (+ x y)))
(f 3 5)
```

application.cmp

```
((env continue)
 (env proc argl continue temp1 temp2 val)
 (proc argl val)
 (
  (assign val (op make-compiled-procedure) (label entry1) (reg env))
  (goto (label after-lambda2))
 entry1
  (assign env (op compiled-procedure-env) (reg proc))
  (assign env (op extend-environment) (const (x y)) (reg argl) (reg env))
  (assign proc (op lookup-variable-value) (const +) (reg env))
  (assign val (op lookup-variable-value) (const y) (reg env))
  (assign argl (op list) (reg val))
  (assign val (op lookup-variable-value) (const x) (reg env))
  (assign argl (op cons) (reg val) (reg argl))
  (test (op primitive-procedure?) (reg proc))
  (branch (label primitive-branch3))
 compiled-branch4
  (assign val (op compiled-procedure-entry) (reg proc))
  (goto (reg val))
 primitive-branch3
  (assign val (op apply-primitive-procedure) (reg proc) (reg argl))
  (goto (reg continue))
 after-call5
 after-lambda2
  (perform (op define-variable!) (const f) (reg val) (reg env))
  (assign val (const f))
  (assign proc (op lookup-variable-value) (const f) (reg env))
  (assign val (const 5))
  (assign argl (op list) (reg val))
  (assign val (const 3))
  (assign argl (op cons) (reg val) (reg argl))
  (test (op primitive-procedure?) (reg proc))
  (branch (label primitive-branch6))
 compiled-branch7
  (assign val (op compiled-procedure-entry) (reg proc))
  (goto (reg val))
 primitive-branch6
  (assign val (op apply-primitive-procedure) (reg proc) (reg argl))
  (goto (reg continue))
 after-call8
 )
)
```