MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall Semester, 1996

**Lecture Notes – November 26, 1996**

## Compilation

If we watched the evaluation of the combination `(F X)`, we would see the following sequence of register operations:

```
(assign unev (op cdr) (reg exp))
(assign exp (op car) (reg exp))
(save continue)
(save env)
(save unev)
(assign val (op lookup-var-val) (reg exp) (reg env))
(restore unev)
(restore env)
(assign proc (reg val))
(save proc)
(assign argl (op empty-arglist))
(save argl)
(assign exp (op car) (reg unev))
(assign continue (label ev-appl-accum-last-arg))
(assign val (op lookup-var-val) (reg exp) (reg env))
(restore argl)
(assign argl (op cons) (reg val) (reg argl))
(restore proc)
;; computation proceeds at apply-dispatch
```

With a compiler, we can build the pieces of the expression directly into the register operations. Thus, we do not need to worry about saving `exp` or `unev`. We can also ignore the continuations generated during evaluation of the pieces of the express (thus we needn't do the initial save of `continue` in the third line above). So simply taking advantage of the fact that the form of the exprssion can be compiled into the flow of the evaluation gives the following alternative to the above code.

```
1.  (save env)
2.  (assign val (op lookup-var-val) (const f) (reg env))
3.  (restore env)
4.  (assign proc (reg val))
5.  (save proc)
6.  (assign argl (op empty-arglist))
7.  (save argl)
8.  (assign val (op lookup-var-val) (const x) (reg env))
9.  (restore argl)
10. (assign argl (op cons) (reg val) (reg argl))
11. (restore proc)
;; computation proceeds at apply-dispatch
```

With more cleverness, we can note some optimizations:

- the instruction in line 2 doesn't clobber `env`, so we needn't save and restore it;
- line 2 might as well move the result directly to `proc`;
- lines 6, 7, and 9 are unnecessary, since the only thing being save and restored is the empty list;
- the argument evaluation doesn't clobber `proc`, so the save in line 5 and the restore in line 11 are unnecessary.

Thus a smarter compiler would generate the following code:

```
(assign proc (op lookup-var-val) (const f) (reg env))
(assign val (op lookup-var-val) (const x) (reg env))
(assign argl (op cons) (reg val) (op empty-arglist))
;; computation proceeds at apply-dispatch
```