6.821 Lecture #8: Non-Hierarchical Scoping and Object-Oriented Programming

October 4, 2007

1 Today
Non-hierarchical Scoping, Object-oriented Programming, HOOK, and HOOPLA.

2 Records
Hierarchical Scoping: Static (Lexical) Scoping and Dynamic Scoping.
Non-hierarchical Scoping: Globals, First-class Environments. For example,

```
(def r1 (record (age 14) (zip 02139)))
(select age r1) → 14
(override r1 (record (boy #t)))
```

These are just examples, we can give this ‘record’ syntax a grammar to guide us:

```
E ::= (record (I E)*) |
     (select (I E)) |
     (override E1 E2) |
     (conceal (I*) E)
```

We also allow the definition of recursive records,

```
(def r2
  (record rec (even? (abs n) (if (= n 0) #t (odd? (- n 1))))
            (odd? (abs n) (if (= n 0) #f (even? (- n 1))))))
((select odd? r2) 3) → #t
```

Make sense? And we allow the use of a tag “with-fields”, which desugars into a series of select statements.

```
(withfields (I1 ... In) E1 E2) →
```

1
(let ((I₁ (select I₁ E₁))
       (Iₙ (select Iₙ Eₙ))
       E₀)
Records, then, are an example of first-class name-spaces. Dave then proceeds to give a couple of examples, which I won’t replicate here. We’re about to use this ‘record’ syntax, to build a simple object-oriented programming language.

3 Object-Oriented Programming

OOP is one way of implementing abstraction. “HOOK” will be our object-oriented language kernel, with the following syntax:

\[
\begin{align*}
P & \in \text{Prog} = (\text{hook}(I*)ED*) \\
D & \in \text{Def} = (\text{def}IE) \\
E & \in \text{Exp} = L|I|(\text{method}M(I_rI_{r*})E_0)|
\quad (\text{compose}E_1E_2)|(\text{null} \rightarrow \text{object})| \\
& \quad (\text{send}ME_1E_{n*})
\end{align*}
\]

And an example:

\[
\begin{align*}
& (\text{def} \text{three}(\text{method} \text{cell}(_\_3)) \\
& \quad (\text{send} \text{cell} \text{three}) \rightarrow 3 \\
& \quad (\text{send} \text{not} \#t) \rightarrow \#f
\end{align*}
\]

etc. A lot of these control-constructs were introduced in SmallTalk. More examples, which I won’t transcribe.

HOOPLA, on the other hand, will be a language (desugared into HOOK) which will define ways of defining classes easily. The “class” element desugars into a pattern of (method new (compose ... |series of methods|)), such that when we send something that we designed with class a “new” message, it returns an object that has all the class methods inside it.

\[
\begin{align*}
& (\text{class}(I*)E*) \leadsto \\
& \quad (\text{method} \text{new}(_\_I*)(\text{object}E*))
\end{align*}
\]

3.1 Inheritance and Multiple Inheritance

We implement inheritance through the an implicit mixing of ‘records’ returned by the send command within a class declaration.

\[
\begin{align*}
& (\text{def} \text{colorpoint}(\text{class}(\text{initxinity clr}) \\
& \quad (\text{send} \text{newpoint initxinity}) \\
& \quad (\text{send} \text{newcolor clr})))
\end{align*}
\]

Make sense?
4 Implementation

How do we implement this, in a language like FL? How do we build objects? What would we do for a method? We can express these with some translation functions:

\[ T[(\text{method} M(I, I^*) E)] = (\text{record}(M(\text{abs}(I, I^*) T[E])) \]

\[ T[(\text{send} M E_0 E_1 \ldots E_n)] = (\text{let}((I, E_0))((\text{select} M I) I T[E_1] \ldots T[E_n])) \]

\[ T[(\text{compose} E_1, E_2)] = (\text{override} T[E_1] T[E_2]) \]

But how do we handle sending messages to literals, as we saw above?

\[ T[N] = (\text{$\text{newinteger}$ N}) \]

\[ (\text{def}$\text{newinteger}$ (\text{abs} (n)) \]

\[ (\text{record}($)\text{val}$ n) \]

\[ (t(\text{abs}(\text{self val})) \]

\[ (\text{override} (\text{$\text{newinteger}$}) (\text{abs} (\text{self val}) \text{self}))) \]

I'm not sure I got that example right, but okay. Dave talks a little more, at this point, about SmallTalk, which took this approach to pretty much every object.