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,

$$(\texttt{def r1}\,(\texttt{record}\,(\texttt{age 14})\,(\texttt{zip 02139})))$$
$$(\texttt{select age r1}) \rightarrow \texttt{14}$$
$$(\texttt{override r1}\,(\texttt{record}\,(\texttt{boy}\,\#\texttt{t})))$$

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

$$\begin{array}{rcl} E & ::= & (\texttt{record}\,(I\,E)*)| \\ & & (\texttt{select}\,(\texttt{I E}))| \\ & & (\texttt{override}\,\texttt{E}_1\,\texttt{E}_2)| \\ & & (\texttt{conceal}\,(\texttt{I}*)\,\texttt{E}) \end{array}$$

We also allow the definition of recursive records,

$$\begin{array}{l} (\texttt{def r2} \\ (\texttt{recordrec}\,(\texttt{even?}\,(\texttt{abs}\,(\texttt{n})\,(\texttt{if}\,(=\,\texttt{n 0})\,\#\texttt{t}\,(\texttt{odd?}\,(-\,\texttt{n 1}))))) \\ \quad (\texttt{odd?}\,(\texttt{abs}\,(\texttt{n})\,(\texttt{if}\,(=\,\texttt{n 0})\,\#\texttt{f}\,(\texttt{even?}\,(-\,\texttt{n 1}))))))) \\ ((\texttt{select odd? r2})\,3) \rightarrow \#\texttt{t} \end{array}$$

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

$$(\texttt{withfields}\,(\texttt{I}_1 \ldots \texttt{I}_n)\,\texttt{E}_r\,\texttt{E}_b) \quad \rightsquigarrow$$

$$
\begin{array}{l}
(\texttt{let}\,((\texttt{I}_1\,(\texttt{select}\,\texttt{I}_1\,\texttt{E}_\texttt{r}) \\
\qquad (\texttt{I}_\texttt{n}\,(\texttt{select}\,\texttt{I}_\texttt{n}\,\texttt{E}_\texttt{r})))) \\
\quad \texttt{E}_\texttt{b})
\end{array}
$$

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:

$$
\begin{array}{rcl}
P \in Prog & = & (\texttt{hook}\,(\texttt{I}*)\,\texttt{E}\,\texttt{D}*) \\
D \in Def & = & (\texttt{def}\,\texttt{I}\,\texttt{E}) \\
E \in Exp & = & L \mid I \mid (\texttt{method}\,\texttt{M}\,(\texttt{I}_\texttt{r}\,\texttt{I}_\texttt{f}*)\,\texttt{E}_\texttt{b}) \mid \\
 & & (\texttt{compose}\,\texttt{E}_1\,\texttt{E}_2) \mid (\texttt{null} - \texttt{object}) \mid \\
 & & (\texttt{send}\,\texttt{M}\,\texttt{E}_\texttt{r}\,\texttt{E}_\texttt{a}*)
\end{array}
$$

And an example:

$$
\begin{array}{l}
(\texttt{def}\,\texttt{three}\,(\texttt{method}\,\texttt{cell}\,(\_)\,3) \\
(\texttt{send}\,\texttt{cell}\,\texttt{three}) \rightarrow 3 \\
(\texttt{send}\,\texttt{not}\,\#\texttt{t}) \rightarrow \#\texttt{f}
\end{array}
$$

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.

$$
\begin{array}{l}
(\texttt{class}\,(\texttt{I}*)\,\texttt{E}*) \quad \rightsquigarrow \\
\qquad\qquad\qquad (\texttt{method}\,\texttt{new}\,(\_\;\texttt{I}*)\,(\texttt{object}\,\texttt{E}*))
\end{array}
$$

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.

$$
\begin{array}{l}
(\texttt{def}\,\texttt{colorpoint}\,(\texttt{class}\,(\texttt{initx}\,\texttt{inity}\,\texttt{clr}) \\
\quad (\texttt{send}\,\texttt{new}\,\texttt{point}\,\texttt{initx}\,\texttt{inity}) \\
\quad (\texttt{send}\,\texttt{new}\,\texttt{color}\,\texttt{clr})))
\end{array}
$$

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:

$$
\begin{aligned}
\mathcal{T}[(\texttt{method}\,\texttt{M}\,(\texttt{I}_\texttt{r}\,\texttt{I}_\texttt{f}*)\,\texttt{E})] &= \\
&\quad (\texttt{record}\,(\texttt{M}\,(\texttt{abs}\,(\texttt{I}_\texttt{r}\,\texttt{I}_\texttt{f}*)\,\mathcal{T}[E]) \\
\mathcal{T}[(\texttt{send}\,\texttt{M}\,\texttt{E}_0\,\texttt{E}_1\,\ldots\,\texttt{E}_\texttt{n})] &= \\
&\quad (\texttt{let}\,((\texttt{I}_\texttt{r}\,\texttt{E}_0))\,((\texttt{select}\,\texttt{M}\,\texttt{I}_\texttt{r})\,\texttt{I}_\texttt{r}\,\mathcal{T}[E_1]\,\ldots\,\mathcal{T}[E_n])) \\
\mathcal{T}[(\texttt{compose}\,\texttt{E}_1\,\texttt{E}_2)] &= \\
&\quad (\texttt{override}\,\mathcal{T}[E_1]\,\mathcal{T}[E_2])
\end{aligned}
$$

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

$$
\begin{aligned}
\mathcal{T}[N] \;=\; & (\$\texttt{newinteger}\,N) \\
& (\texttt{def}\,\$\texttt{newinteger}\,(\texttt{abs}\,(\texttt{n}) \\
& \quad (\texttt{record}\,(\$\texttt{val}\,\texttt{n}) \\
& \qquad (\texttt{t}\,(\texttt{abs}\,(\texttt{self}\,\texttt{val}) \\
& \qquad\quad (\texttt{override}\,(\$\texttt{newinteger} \\
& \qquad\qquad (@\texttt{t}\,\texttt{n}\,(\texttt{select}\,\$\texttt{val}\,\texttt{arg})))\,\texttt{self})))
\end{aligned}
$$

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.