1. Course basics

1.1. Goals.
- Tools for understanding and writing disciplined programs.
- Three units: types, abstract interpretation, and model checking.
- All three are logically equivalent; have trade-offs (how conservative they are, how they perform, etc.) for achieving the same thing.

1.2. Resources.
- Course website: http://stellar.mit.edu/S/course/6/fa10/6.820/
- Textbook: *Types and Programming Languages*, by Benjamin Pierce.
- Papers from syllabus.
- Staff mailing list: 6820-staff@lists.csail.mit.edu.
- Teaching assistant: jeanyang@csail.mit.edu.

2. Functional Programming

2.1. Basics.
- Functions
  \[ \lambda x. x \]
  \[ \lambda x. x + 1 \]
  \[ \lambda f. \lambda x. f \ x \]
- Function application
  \[ (\lambda x. x) \ 1 \]
  \[ (\lambda x. x + 1) \ 0 \]
  \[ ((\lambda f. \lambda x. f \ x) \ (\lambda x. x + 1)) \ 0 \]

2.2. Some vocabulary.
- Reduction (call-by-name vs. call-by-value)
- Normal form
- Lexical scoping

*Date: Friday, September 10, 2010.*
3. Haskell: FP + Types

3.1. The Haskell language.

- Purely functional (effects using monads)
- Strongly statically typed
- Type inference via Hindley-Milner algorithm
- Call by need with \textit{thunks}

3.2. Advantages of using Haskell.

- Concise

\begin{verbatim}
qsort [] = []
qsort (p:xs) = qsort lesser ++ [p] ++ qsort greater
\end{verbatim}

\begin{verbatim}
where
  lesser = [ y | y <- xs, y < p ]
  greater = [ y | y <- xs, y >= p ]
\end{verbatim}

- Enforces programming discipline
- Has really cute types