Common LISP

- A combination of many of the features of the popular dialects of LISP around in the early 1980s
- A large and complex language--the opposite of Scheme
- Includes:
  - records
  - arrays
  - complex numbers
  - character strings
  - powerful i/o capabilities
  - packages with access control
  - imperative features like those of Scheme
  - iterative control statements

Standard ML

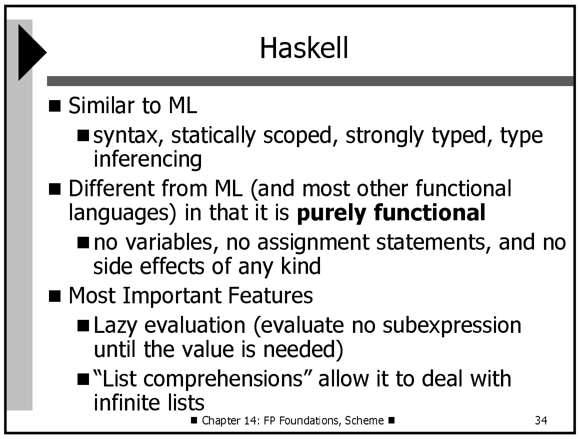
- Statically scoped, with syntax closer to Pascal than LISP
- Uses type declarations, but also does **type inferencing** to determine the types of undeclared variables (See Ch. 4)
- Strongly typed (whereas Scheme has latent typing), no type coercions
- Includes exception handling and a module facility for implementing ADTs
- Includes lists and list operations
- The **val** statement binds a name to a value (similar to DEFINE in Scheme)

SML Function Declarations

fun *function_name* (*formal_parameters*) =
function_body_expression;

```
fun cube (x : int) = x * x * x;
```

- List-based operations can be polymorphic, using type inferencing
- Functions that use arithmetic or relational operators cannot be polymorphic



Haskell

- Similar to ML
  - syntax, statically scoped, strongly typed, type inferencing
- Different from ML (and most other functional languages) in that it is **purely functional**
  - no variables, no assignment statements, and no side effects of any kind
- Most Important Features
  - Lazy evaluation (evaluate no subexpression until the value is needed)
  - "List comprehensions" allow it to deal with infinite lists