1 11

1. Scheme
   1. Eq/2- same pointer value
   2. Eqv- operational equivalent
   3. Equal- same structure and contents
   4. Sort- sorts a list
   5. Merge- merges 2 lists
2. Lambda calculus
   1. Goals
      1. Functional language
      2. Pure lambda calc
      3. Build tm equivalent programming language
      4. Conditionals, Booleans, integers
      5. Recursions and arithmetic
      6. Types and data structures
      7. Evaluation schemes
   2. As powerful as a turing machine
   3. No numbers, no true/false, no and/or, just functions
      1. Functions of what? Uninterpreted till given semantics
   4. Functional language
      1. Programs as parameterized problem instances
      2. Abstract versions- 3+4 => plus(3 4)
      3. Data type- functions
      4. Names are only formal parameters, given values by function calls with them as formal parameters
   5. Execution order
      1. None really
      2. One giant nested function
   6. No global variables
   7. Explicit representations of data structures
      1. Have to pass in whole data structures
      2. FUNCTIONS AS VALUES
   8. Theory of computation
      1. Church turing hypothesis- computable fn = computable by turing machine
      2. Church rosser therom- some lambda expressions might not terminate
         1. Some might terminate when called by name but won’t terminate when called by value
            1. Call by value- (+ 3 4)
            2. Call by name-(+ a b)
   9. Pure lambda calculus
      1. Functionalization
      2. Application
      3. Everything is a function
   10. Abstraction and specialization
       1. Abstraction- turn expression into a function by introducing a parameter
       2. Specialization- replace name with value
   11. Lambda expressions
       1. One of 3 things
          1. Name
          2. Function
          3. Application of function
       2. <function> :: lam <name>.<body>
       3. <Body > :: <expression>
       4. Examples
          1. lX.x
          2. l first. L second. First
          3. lf. la. (f a)
       5. lambda comes before formal parameter (name used for abstraction)
       6. “.” Separates it from body
       7. Applications
          1. Parentheses
          2. Function expression
          3. Argument expression
          4. (fe ae)
          5. Identity function
             1. (lx. X Joey)- returns Joey
          6. Produces bound pair
             1. Name
             2. Argument
             3. Value that gets substituted