CSci 450: Organization of Programming Languages
Exploring Languages using Interpreters and Functional Programming

H. Conrad Cunningham

16 April 2018

Contents

Acknowledgements: I adapted and revised much of this work in Summer and Fall 2016 and Spring 2017 from my previous materials. I added new materials in Spring and Summer 2017. This is an ongoing effort that will continue through at least Fall 2018.

In Spring 2018, I linked in modified Programming Paradigms and Abstraction documents that will be included in a future draft of this material. I also changed to my new working title for this work.

I maintain these notes as text in Pandoc's dialect of Markdown using embedded LaTeX markup for the mathematical formulas and then translate the notes to HTML, PDF, and other formats as needed.

Advisory: The HTML version of this document requires use of a browser that supports the display of MathML. A good choice as of April 2018 is a recent version of Firefox from Mozilla.

Exploring Languages using Interpreters and Functional Programming

Chapter 0: Course Introduction

- **Not updated for Programming Languages course** [HTML] [PDF]

Chapter 1: Fundamental Concepts

- [HTML] [PDF]
- Partially reorganized material in Spring 2018 (I plan to break old chapter 1 into two chapters and an appendix. This is a new draft of the second part.)
  - Programming Paradigms (drawn from old sections 1.3, 1.4, 1.7) [HTML] [PDF]
- Supplementary slides:
  - Evolving Computer Hardware Affecs Programming Languages (HTML)
  - Primary Programming Paradigms (HTML)
  - History of Programming Languages (HTML)

Chapter 2: Basic Haskell Functional Programming

- [HTML] [PDF]
- Partially reorganized material in Spring 2018 (I plan to break old chapter 2 into two chapters. This is a draft of the second part.)
  - Abstraction (drawn from old sections 1.6, 2.5, 2.6, 2.7) [HTML] [PDF]
- Supplementary slides:
  - Our First Haskell Functions (HTML)
  - Top-Down Stepwise Refinement (HTML)
  - Using Data Abstraction (HTML)

Chapter 3: Evaluation and Efficiency

- [HTML] [PDF]
- Supplementary slides:
  - Evaluation of Functional Programs (HTML)
  - Recursion Styles (HTML)

Chapter 4: List Programming

- [HTML] [PDF]
- Supplementary slides:
  - List Programming (HTML)
- Supplementary notes used Spring 2017
  - Candy Bowl ADT Semantics
- Possible future material on modular programming based on:
  - Lecture Notes on Modular Design
  - Lecture Notes on Data Abstraction
  - Labelled Digraph Abstract Data Type
  - Cookie Jar ADT Problem Description (Scala)

Chapter 5: Higher-Order Functions

- [HTML] [PDF]
- Supplementary slides:
  - Higher-Order List Programming

  - Haskell Function Concepts
- TBD: Regular expression combinators

Partial Chapter 6: Developing Functional Programs

- [HTML] [PDF]

Partial Chapter 7: More List Processing and Problem Solving

- [HTML] [PDF]

Partial Chapter 8: Algebraic Data Types

- [HTML] [PDF]
- Supplementary slides:
  - Algebraic Data Types

Partial Chapter 9: Type Classes

- [HTML] [PDF]
- Supplementary slides:
  - Overloading and Type Classes

Chapter 10: Expression Language Syntax and Semantics

- [HTML] [PDF]
- Modules
  - Abstract Syntax module
  - Evaluator module
  - Environments module
  - Values module
  - Process AST module (in work) (simplfication and derivation on ASTs)

Chapter 11: Expression Language Parsing

- [HTML] [PDF]

  Supplemental slides:

  - Expression Language Parsing

- Common modules (Chapters 10 and 11)
  - Lexical analyzer module
  - Abstract Syntax module
  - Evaluator module
  - Environments module
  - Values module
- Prefix syntax specific modules
  - Prefix REPL module
  - Recursive descent parser for prefix expression language
  - Test Prefix (in work, not current)
  - Prefix REPL module
- Infix syntax specific module
  - Infix REPL module
  - Recursive descent parser for infix expression language (simple expressions)
  - Test Infix (in work, not current)
- Other modules
  - Parser combinators module (in work)
  - Test parser combinators (in work)
  - Process AST module (in work) (simplfication and derivation on ASTs, chapter 10)
  - Stack Virtual Machine module (in work) (code generation, chapter 10)
  - Parser Combinators module

Partial Chapter 12: Expression Language Compilation

- [HTML] [PDF]
- Code in work
  - Stack Virtual Machine?

Chapter 13 in Work: Imperative Core Language

- Imperative Core Language Case Study (does not exist yet as of 27 Sep)
- Interpreter modules (prefix syntax) code mostly works but needs a bit of update to match recent changes to Expression Language
  - REPL module
  - Recursive descent parser module
  - Lexical analyzer module
  - Abstract Syntax module
  - Evaluator module
  - Environments module
  - Values module
  - Test Imp Core(in work, not current)

Future Chapters? More Languages and Interpreters

- TBD: Add function definitions and calls, local variables
- TBD: Simple Imperative Core Language
- TBD: Simple Lisp/Scheme language
- TBD: Modular/OO language
- TBD: Lazy language
- TBD: Compilation
- TBD: Continuations

Future Chapters? Using Algebraic Data Types

- TBD: Error handling with `Maybe` and `Either` – Handling Errors Without Exceptions (Scala)
- TBD: Regular Expressions using algebraic data types
- Not used Spring 2017
  - Framework Design Using Function Generalization: A Binary Tree Traversal Case Study
  - Mealy Machine Simulator
  - Labelled Digraph Abstract Data Type

Partial Chapter: Domain Specific Languages

- Domain Specific Languages
- Sandwich DSL Case Study

  - Haskell version
  - Scala version
  - old Lua version
- DSL in other languages (not used Spring 2017)
  - State Machine DSL (Scala)
  - Lair DSL (Lua)
  - Survey DSL (Ruby)
  - Computer Configuration DSL (Scala)
  - Email DSL (Scala)

Future Chapter? Input/Output

Future Chapter? Testing

Partial Chapter: Algebras, Laws, and Synthesis

- TBD: Discussion of functors, applicative functors, and monads (perhaps semigroups and monoids) – perhaps use `Maybe`
- From *Notes on Functional Programming with Haskell*:
  - Chapter 11, Haskell Laws
  - Chapter 12, Program Synthesis

Partial Chapter: Reduction Models

- From *Notes on Functional Programming with Haskell*:
  - Chapter 13, Models of Reduction
  - Chapter 14, Divide and Conquer

Partial Chapter: Infinite Data Structures

- Strictness and Laziness (Scala)
- From *Notes on Functional Programming with Haskell*:
  - Chapter 15, Infinite Data Structures

Future Chapters? Games

- Wizard's Adventure game (Elixir)
- Dice of Doom (Elixir)

Future Chapter? Concurrency

- TBD

Future Chapter? Parallelism

- TBD

Future Chapter? Reactive Programming

- TBD