Chapter 0

Prelude

Computational Processes

- Abstract beings that inhabit computers
- Manipulate data
- Directed by a program
- Written in a programming language

The Tool of this Course: Scheme

- Dialect of Lisp (1958)
- Proposed in 1975
- Extremely powerful and elegant
- Standardized into R^{n}Rs

R6Rs

- Many implementations available

I use DrRacket

- Allows you to “go meta”

actual goal of this course

Study Material

- Chapters 1, 2, 3, 5, 6: Structure and Interpretation of Computer Programs (Gerald Jay Sussman and Hal Abelson): chapters 1, 2, 3, 4
- Chapters 4, 7: Slides + notes in classroom

Chapter 1: Fundamentals of Higher Order Programming

1. Scheme S-expressions, Function definitions
2. Lexical Scoping vs. dynamic scoping
3. Iteration as Optimised Tail Recursion
4. Higher Order Procedures and Anonymous Lambda’s.

Chapter 2: Advanced Higher Order Programming

1. Cons-cells, lists and nested lists.
2. List processing and Higher Order List Procedures
3. Symbols and Homoiconicity: Quoting Lists
4. Homoiconicity for Meta-programming
5. Case Study: Symbolic derivation

Chapter 3: Fundamental Concepts of State, Scoping and Evaluation Order

1. begin, set! and mutable state
2. Objects as closures
3. Environment diagrams, box-and-pointer diagrams
4. (Infinite) streams and lazy evaluation.
5. delay and force.

Chapter 4: Continuations and current-continuations

1. Continuations
2. call-with-current-continuation
3. An implementation of
   1. goto,
   2. yield,
   3. coroutines
   4. exception handling

Chapter 5: Semantics of Higher-Order Languages

1. Concrete vs. Abstract Syntax
2. Meta circular interpretation
3. The analysing interpreter (i.e. compiler)
4. CPS interpretation and semantics of call-with-current-continuation

Chapter 6: Variations on the Semantics

1. A lazy evaluation version of Scheme + thunkified interpreter
2. A nondeterministic version of Scheme + continuation-based interpreter

Chapter 7: Introduction to the λ-calculus

1. λ-expressions and β-reduction
2. Computability in λ-calculus:
   a construction of functional programming languages
3. Recursion and the Fixed-point Theorem.