CSE 505: Programming Languages (Fall ’98)

Instructor: Craig Chambers

- office hours: TuTh 11:00-12:00 (Sieg 309)

TAs: Vass Litvinov and Jonathan Aldrich

- Vass Litvinov’s office hour: Tuesday 1:30-2:30 (Sieg 225)
- Jonathan Aldrich’s office hour: Friday 2:30-3:20 (Chateau 109a)

Goals:

- Understand the major concepts in programming languages
- Gain some experience using advanced features of modern programming languages
- Learn to use formal semantics as a tool for understanding and describing language semantics precisely
- Become aware of current open problems and active research areas in the design of modern programming languages
- Understand trade-offs in language design

Readings:

- *Advanced Programming Language Design*, Finkel
- copies of overviews of languages being studied
- important papers from the literature

Grading:

- homework assignments: 15%
- course project: 20%
- two tests: 20% each
- final exam: 25%

Course outline

Functional languages

- Scheme
  - atoms, symbols, lists, cons, car, cdr; define, let, let*, cond
  - higher-order first-class functions
  - tail recursion
  - eval
  - continuations
- ML
  - lists, tuples; val, fun, if, let
  - pattern matching
  - type inference
  - polymorphic types
  - higher-order functions
  - datatypes, polymorphic datatypes
  - exceptions
  - references
  - modules; signatures and structures
- Haskell
  - lazy evaluation
  - array & list comprehensions
  - type classes

Object-oriented languages

- Smalltalk
  - objects, classes, instances, methods, instance variables
  - inheritance
  - dynamic dispatching
  - first-class classes; meta-classes
- Cecil
  - multi-methods
  - multiple inheritance
  - subtyping; subtype polymorphism
  - inheritance vs. subtyping
  - covariance vs. contravariance
  - bounded quantification; F-bounded quantification; matching

Declarative languages

- Prolog
  - clauses
  - unification
  - backtracking
  - cut
- CLP(R)
  - constraints

Formal semantics

- lambda calculus
    - untyped lambda calculus
    - alpha, beta, eta reduction; normal order vs. applicative order evaluation; confluence, normalization
    - simply typed lambda calculus
    - polymorphically typed (predicative and impredicative) lambda calculus
- denotational semantics
    - lattice domains
- operational semantics
    - inference rules
    - big step/natural semantics
    - small step semantics
- axiomatic semantics