Advanced Programming Languages

Daily Topics

Day 1:

- Overview of Advanced Programming Languages
  - Elements of Programming Language Design
  - Abstractions in Programming
- Language Semantics (domain) ⇨ Program style ⇨ Software Quality
  - Common abstractions at basis of all PLangs
- Importance and value of FP
  - Basis in mathematics → λ_c
  - Correctness, Abstraction, HLL
- Formal Semantics
  - Language by example ⇨confusions
  - Informal (≈*English*) ≠ precise, clear, simple
    - ⇨ long, complex, obfuscated…
  - Axiomatic, Operational, Denotational, …
- Course Topics
  - FP → SML
  - PLang model(s)
  - Denotational Semantics
  - Semantic prototyping (in SML)
- Course Structure & Administration
  - Daily Schedule
  - Quizzes
  - Labs
  - Grading, Exams
- Reading
  - Homework

Day 2: Syntax & Grammar

- Review…
- Programming Language Families & Paradigms
  - IP, PP, LP, CP, DP, OO, FP, …
  - Language families, evolution, & paradigms
  - Multi-paradigm..?
  - Functional, Imperative, Scripting, …
- Features of FP
  - Non-destructive ⇨ *referential transparency*
  - FCF, HOF ⇨ functional composition
  - True-polymorphism, type inference, patterns, …
- Syntax Definition ⇨ Grammar
  - Formally: G = < Σ, N, P, S >
  - Chomsky Hierarchy = Machines :: Languages
  - Meta-language = BNF
    - Self-defining? $L_1 \leftarrow$ BNF $\leftarrow L_2 \leftarrow L_3 \leftarrow L_4 \ldots \Rightarrow .. D_D$
  - Parsing tools and process
- Levels of Syntax
  - Micro: lexical tokens
  - Macro: Phrase structure
  - Parsing & Lexical tools
- Vedic Language: Name ≡ form
- Reading :*Backus*
  - Homework: Three summary points from reading

Day 3: Semantic Models and Principles

- Quiz
- Review…
  - Backus
  - IP ⇨over-constrained (=*how*) ⇨ detailed, LLL, sequential
  - FP ⇨ HLL *(=what*) ⇨ abstraction, composition, …
- Basic (semantic) Elements of programming languages
  - Review
  - Identify basic semantics of program(s)
- Semantic Definitions
  - Semantic domains
    - values
  - Semantic Model
    - Store, environment
  - Semantic elements
    - Commands, Definitions, Expressions
  - Semantic equations
    - C(Cmd) → Δ Sto
    - D(Def) → Δ^+ Env
    - E(Exp) → Value
- Composition, and constructs
  - Qualification, composition
  - D{D}, D{C}, D{E}, …
- Simple semantic (design) principles
  - Type completeness
- Semantic principles
  - Principle of qualification
    - D{C} → D, … ⇨ D{ S_i } → S_i
  - Principle of abstraction
    - F_C ( ) → C, … ⇨ F_{Si} ( ) → S_i
  - … (more later)

Day 3b: Introduction to FP in SML

- Note changed model for FP
    - No Cmd, so only {Def, Exp}
    - No S.E.
- Main characteristics of FP
- Tools for SML
- Reading: Michaelson
    - Homework (Main Points from reading)

Day 4a: Semantic principles and concepts

- Quiz & review
- Semantic Principles: (Review)
  - Principle of Type Completeness
  - Orthogonally
  - Abstraction
  - Qualification
  - Correspondence
  - Parameterization
- Declarations & Composition
  - Sequential, Simultaneous, Qualification, Recursive
- Use semantic model to understand Java/IP features
- First Class Functions (FCF)
  - *First-Class* = All (appropriate) operations
  - FCF → pass, return as values
    - basis of HOF
- Lambda λ
  - Standard model for function abstractions
  - Anonymous function abstraction
  - ≈ function expression
- Closures
  - Important –basis of FCF
  - Java ≈ inner classes
  - Issue: HOF connection to enclosing environment(s)
    - Stack based ARs
  - Downward FunArgs ≈ OK
    - Pascal
    - Requires closures
  - Upward FunArgs
    - Harder,
    - Lifetime(closure) > lifetime(stack based AR)
  - Partial application

Day 4b: Introduction to Functional Programming & SML

- Review…
  - Have simple semantic model of IP
- Note definitions of FP
  - No ***Cmd***, so only {***Def***, ***Exp***}
  - HOF, FCF
  - Currying, partial application
  - Type inference
  - Patterns and unification
  - (true) parametric polymorphism
  - Type completeness
- SML ∈ FP
  - SML: modern, new, different(!), powerful, simple
- More examples: *day1.ml*
  - Semantic features
  - In SML syntax…
  - Values
    - ***Val*** ∈ Def (≠ Assign, Cmd)
    - non-destructive → referential transparency
  - Functions, HOF
  - Type signatures, type inference
  - Closures & Partial applications
  - Some non-pure FP features; I/O
- Michaelson examples
- Reading & Assignment: Watt §13

Day 5: Features of FP in SML (Review)

- (From review of day1.ml)
- Semantic Principles: (Review)
    - Referential transparency
        - No assignment /cmds
        - Only definitions
        - Incremental nested scopes
    - (auto-) type inference & signatures
        - fn : int → int
        - Function types are right-associative
    - Lambda
        - = Anonymous function expression
        - ∈ FCF
    - Functions
        - Always one argument (at a time…) (!) (monadic)
            - Tuples: multiple → one
        - fn : int → int → int
        - Curried functions ⇨ HOF
        - Partial application
        - Function application binds tightly, is left-associative
    - Polymorphism
        - Type variables α (‘a)
            - = type *Type* // meta-type!
    - Patterns
        - Equational style
    - Other…
        - Type constraints (:real)
        - Unit ()
        - No auto-conversions (strict types)

Day 5: Functional Programming & SML

- Review of day2.ml examples…
  - Curry/ PA → incremental specialization, re-use
  - $add|_1$ (1) → $increment|_1$
- Compare state in Fp
  - Recursion vs. iteration
  - Bindings v.s. store
- Polymorphism
  - Any type ⇨ type insensitive
    - Can have limited poly: $\alpha^{=}$ (??)
  - Lists…
  - Parametric (*real*), not *ad-hoc*
- ML features
  - Patterns → { match, unify)}
  - Tuples :: heterogeneous, ordered
  - Lists :: homogenous
- List processing
  - hd, tail ⇨ pattern match (x::Xs)
  - constructor **::**, append @
    - constructors ⇨ create (unify), and decode
  - generally use polymorphism & recursion
  - Recursion & patterns
- Definitions
  - Recursive : *rec*
  - Functions: $\lambda|_1$ or $fun|_n$
- FP idioms
  - Functional composition: o
  - Helper functions
- Examples…
- Assign: Watt & Michaelson; reading, exercises

Day 5: Review: Michaelson

- § 9.1 Types
- § 9.3 Basic Types
  - bool, string, int
- § 9.4 Lists
  - Polymorphic, homogenous, variable length
- § 9.5 Tuples
  - Heterogeneous, ordered, fixed length
  - Selection – bind name to field(s)
    fun get3rd (_,_,n:int) = n;
- Strings
  - Standard operations: ^ and or + * …
  - List operations **::** hd tail size
- Pattern matching
- Type expressions = *alias*
  - Defines new types
  - Gives name to a structural pattern
    - type checking on usage
  - Type constructors: * + →
- Datatypes
  - User defined type and constructors (and fields)

Day 6: Functional Programming & SML: Higher Order Functions

- Review…
  - Review Watt reading & solutions
- Continue with *Day2.ml* examples…
- User defined types
  - Type constructors
- ML features
  - Patterns over { constructors, constants)
    - ≈ inverse OO
    - Function dispatched *ad-hoc polymorphism*
  - Constructors = duality ⇨ construct, match+unify (de-compose)
- Compare patterns : Polymorphism
  - FP = parametric
  - OO = sub-type
  - FP patterns ≈ overloading → ad-hoc poly
- Examples & features
  - op & $op
  - composition: o (monadic), oo (dyadic)
    - combinators
    - sections { *secl, secr* }
- FP idioms
  - filter, map, reduce
    - very general; most functions in terms of these
    - reduce = fold ⇨ foldl/foldr
    - separates operation from control (=recursion)
    - most general = encapsulates general (*primitive recursive*) recursion
      - can so fold → {filter, map, reduce} → …. ∞
  - accumulators
    - → tail-recursion

Day 7: Functional Programming & SML: Examples and methods

- Quiz & Review;
  - Review homework exercises (any, all, member, …)
- Examples & features (Day2, Day3, …)
  - Nested λ expressions ⇨ multiple args.
  - op & $op
  - composition: o (monadic), oo (dyadic)
    - combinators
    - sections { *secl, secr* } ≈ non-commutative PA
      - *position selective PA composition*
- Eager /lazy evaluation
  - ML functions are *applicative (eager) = CBV*
    - alternate: *normal (lazy)*
    - eager ⇨ strict
  - two special forms
    - **andalso** (≡ if/then/else) & **orelse**
    - "*short-circuit*" = lazy
- FP idioms
  - curry/uncurry
    - goals, definitions, and signatures
    - conversion examples
- Other ML features /issues
  - equality polymorphism: a', a'' (α, $\alpha^{=}$)
  - top-level value polymorphism.. (not!)
- Review Michaelson reading & solutions
  - Types, constructors
  - Review Watt reading & solutions
- Next:
  - Review and Summary of FP & SML

Day 8: Review & Summary of FP

- Quiz & Review;
- Review Michaelson reading & solutions
    - Types, constructors
    - Expressions
        - Interpreter: N → n –(computation)→ n' ⇨ N'
        - NB: *Numeral* ≠ Number !!
    - Review Michaelson & Watt reading/solutions
        - Details of solution: 9.6
- Data structures
    - SR, but no (visible) pointers!
    - Examples (Michaelson 9.18)
        - Lists (and DIY lists)
        - Trees
    - Constructive tree mappings (Watt.1 - 13.1.2)
- trees.ml
    - dictionary: functional data structure
- Exercise(s)
    - Write signature & definition for *dict* example
- Next:
    - Introduction to Denotational Semantics

Day 9: Introduction to Denotational Semantics

- Quiz & Review
- Goal is formal model
- Types of models
    - AS, DSem, OS, … AS, …
- Standard elements and format for models
    - Syntactic domains
    - Syntactic equations
    - Semantic domains
    - Semantic functions
        - Declarations
        - Definitions
- Simple (standard) semantic model
    - Store, environment
- Binary Numerals example (Ex. 3.1)
    - Maps: Numerals ⇨ Numbers
- Calc.1 (Ex. 3.2)
    - Maps: Keystrokes ⇨ Numbers
- Model of Store
    - Functional model (HOF dictionary)
        - Simple, (≠ efficient)
        - NB: models store in FP (no store!)
    - Direct and indirect denotations

Day 10: Denotational Models: Stores

- Quiz & Review
- Semantic Model^{++}
- Model of Store
    - Functional model (HOF dictionary)
        - Simple, (≠ efficient)
        - NB: models IP store in FP (no store!)
          (Represent change from within non-change)
    - Tagged locations
      { Storable, unused, *undef* }
    - Fancier model with memory management (*allocate*, *deallocate*)
      (for later…)
- Now can model Semantics with ***Commands***
    - Calc.2 (Ex. 3.3)
    - New additional arguments to semantic equations
      ⇨ model state (Store)
    - Review details of semantic equations

Day 11: Denotational Models: Environments

- ~~Quiz~~ & Review
- Semantic Model^{++}
- Semantic Model
  - Definitions, blocks
  - Scope + Defn → Binding(s) → Env
- Model of Environment
  - Direct and indirect denotations
  - Domains and types { *undef*, Denotable }
- Implementation
  - Functional model (HOF dictionary)
- Semantics with Definitions
  - Calc.3 (Ex. 3.3)
  - New additional arguments to semantic equations
    ⇨ model Definitions (Environment)
  - Review details of semantic equations
- Review for exam (?)

Day 12: Denotational Models: Stores & Environments

- ~~Quiz~~ & Review
- Semantic Model^{++}
  - Combine { Sto & Env }
  - ⇨ { Exp, Cmd, Def } ☺
- Semantic Model
- Definitions
  - Review Imp: (Watt Ex. 3.6)
  - Loop → Recursion (!)
- Semantic empathy (of meta-language)
  - Recursive Definitions
  - Lazy construct(s)

Day 13: Denotational Models: Function Abstractions

- Semantic Model^{++}
  - Combine { Sto & Env }
  - ⇨ { Exp, Cmd, Def } ☺
- Structure and semantics of function definitions
  - Name (Parameters) {Body}
  - Parameters ⇨ local environment(s) ($e_d + e_p$)
  - Arguments: evaluate in e_i (invocation)
- Recall; *Principle of Correspondence*
  - Argument passing ← → local environment
- And other *Semantic Principles*
  - f(){ S_i } → S_i *Principle of Abstraction*
  - D{ S_i } → S_i *Principle of Qualification*
  - ∀ S_i → f(S_i) *Principle of Parameterization*
- Abstractions (∀ S_i …)
  - Cmd → procedure
  - Expr → function
  - Declarations → module (class)
  - Type → class
  - L-value → ??
  - Location → L-value
- Implementation
  - (Args ∈ RT) → (e_p ∈ CT) → Eval{ body }
- Definitions
  - Review Exp.1: (Watt Ex. 3.7)
  - Recursion(?) (not yet..)
    - f ∉ e_d
  - Review Imp.1: (Watt Ex. 3.8)

Day 14: Denotational Models: Function Abstractions & Parameters

- Semantic Model^{++}
  - [function] Abstracts
  - Review: Exercise
    - Write the definitions from Exp.1 for functions
- Other parameter/argument types?
  - R-values$|_{value}$ → CBV
  - L-values$|_{loc}$ → CBR
  - Many other "call-by"s …
- Definitions
  - Review: Exp.3 (Watt Ex. 3.10a)
    - Single parameter
  - New: Exp.4 (Watt Ex. 3.10b)
    - Multiple parameters
- CBV/CBR Exercise
  - Which examples work?
  - Use: logic/experience, principle of correspondence, Equations (to prove)
- Recursive abstractions (Exp.2)s
  - Recursive? → ML recursion (!)
  - Semantic empathy (of meta-language)
- Review readings & Homework's

Day 15: Denotational Models: Language Definitions

- Semantic Models
  - Domains of semantic functions
- Example: (4.1)
  - Expressions with S.E.
    - { C; E } → E (??!) ☹
    - { C, E } → C ✓ Still ☹
    - ⇨ Neither!
  - Forces redefinition of ***all*** Expressions
  - Also forces evaluation order → non-logical, constrained
  - Typical example: I^{++}
    - ⇨ Value + Sto'
  - Thus programs harder to understand & reason about
- Structure of Programs
  - Syntactic Domain
    - Program ::= **program** (**input** id : T, **output** Id : T) ~ Cmd
  - And Semantic equation
    - Run: Program → value$|_{in}$ → value$|_{out}$
    - + Definition…
- I/O in programs
  - I/O is not functional
  - Not logical, sequential, side effect
  - Can model as *stream*
  - Or ***Monad***

Day 16: Denotational Models: Language Prototyping

- From abstract to concrete
  - Convert pseudo-ML → SML
- Example: (4.1)
- Introduction to Prototyping lab(s)
  - Basic skeleton given
  - Complete definitions
  - Choose extensions
- Evaluate the semantics of the Target language (~IMP)
  - Closures?
  - HOF?
  - Recursion?
  - Eager/lazy?
- Semantic Prototype Lab
  - Start ≈ Ex. 4.1
  - Lab Phases…
  - Timesheet
  - Policies…

Day 17: Continuations: An abstraction & model of control

- Semantic Model:
  - Model of control?
  - So far; unspecified (**Expr**)
  - Or sequential (**Cmd**) [Why?]
  - Yet, several missing elements
- Error handling
  - Now in ML implementation, but not language definition
  - Implied sequential propagation
  - Messy to add to descriptions,
    can simplify with better composition ★
  - Still, is error chaining; sequential propagation
  - ≠ exceptions, which are global handling
- Continuations
  - Convert from sequential parts ⇨ wholeness
  - Are programming technique (language feature)
  - Good semantic model
  - Good model and tool for implementation (compilers)
  - Others: backtracking, threads, events, errors, exceptions, co-routines, …
  - Current usage: **return** ≈ *anon (implicit)continuation*
- Semantic Model^{++}
  - ***CCont, ECont, DCont***
    - DSem become incremental semantic compositions
    - Semantic elements as continuation transformers
  - CPS – make continuations explicit, and manipulatable
  - Affects all equations (!) → new semantic arguments
  - Naturally leads to functional (continuation) compositional forms
- Model^{++} → Features^{++}
  - Return, jump, exceptions, …
  - Control abstracts → *sequencers*
- Assignment:
  - Read Tennent §13.1-3

Day 18: Continuations: Continuation Semantics

- Adding continuations to semantic model
  - ***CCont, ECont, DCont***
  - Revised domains
  - Semantics become *Continuation transformers*
- This is the *standard model*
  - Example of typical equations
- Read Tennent definitions
  - Review (13.2) – basic notation
  - T13.3: ≈ same as our IMP model
  - T13.4 ⇨ Continuation semantics
- Assignment:
  - Read Tennent §13.4,7

Day 19: Continuations: Review

- Review basic equations
- Exercise:
    - Write equations…
- Additional features
    - goto
    - leave (ex. 13.
- Review tests in *Impc*
- Assignment:
    - Lab: *Impc.ml*