Elements of Programming Languages

Course review

James Cheney

University of Edinburgh

November 27, 2017

Overview

- We've now covered
  - Basic concepts: ASTs, evaluation, typing, names, scope
  - Common elements of any programming language
  - Programming in the large: components, abstractions
  - Language design issues
- Today:
  - Review of course, pointers to related reading
  - Information about the exam
  - Conclusions

Intro & Abstract syntax

- Concrete vs. Abstract Syntax
- Abstract syntax trees
- Abstract syntax of L_{Arith} in several languages
- Structural induction over syntax trees
- Reading: PFPL2 1.1; CPL 4.1, 5.4.1

Evaluation & Interpretation

- A simple interpreter for arithmetic expressions
- Evaluation judgment $e \Downarrow v$ and big-step evaluation rules
- Totality, uniqueness, and correctness of interpreter (via structural induction)
- Reading: PFPL2 2.1-3, 2.6, 7.1, CPL 5.4.2

Booleans, conditionals, types

- Boolean expressions, equality tests, and conditionals
- Typing judgment $\vdash e : \tau$
- Typing rules
- Type soundness and static vs. dynamic typing
- Reading: PFPL2 4.1-4.2, CPL 5.4.2, 6.1, 6.2

Variables and scope

- Variables: symbols denoting other things
- Substitution: replacing variables with expressions/values
- Scope and binding: introducing and using variables
- Free variables and α-equivalence
- Impact of variables, scope and binding on evaluation and typing (using `let`-binding to illustrate)
- Reading: PFPL2 1.2, 3.1-3.2, CPL 4.2, 7.1

Functions and recursion

- Named (non-recursive) functions
- Static vs. dynamic scope
- Anonymous functions
- Recursive functions
- The function type, $\tau_1 \to \tau_2$
- Reading: PFPL2 8, 19.1-2; CPL 4.2, 5.4.3

Data structures

- Pairs and pair types $\tau_1 \times \tau_2$, which combine two or more data structures
- Variant/choice types $\tau_1 + \tau_2$, which represent a choice between two or more data structures
- Special cases `unit`, `empty`
- Reading: PFPL2 10.1, 11.1, CPL 5.4.4

Records, variants and subtyping

- Records, generating from pairs to structures with named fields
- Named variants, generalizing from binary choices to named constructors (e.g. datatypes, case classes)
- Type abbreviations and definitions
- Subtyping (e.g. width subtyping, depth subtyping for records)
- Covariance and contravariance; subtyping for pair, choice, function types
- Reading: CPL 6.5; PFPL2 10.2, 11.2-3, 24.1-3

Polymorphism and type inference

- The idea of thinking of the same code as having many different types
- Parametric polymorphism: abstracting over a type parameter (variable)
- Modeling polymorphism using types $\forall A.\tau$
- High-level coverage of type inference, e.g. in Scala
- **[non-examinable]** Hindley-Milner and let-bound polymorphism
- Reading: PFPL2 16.1; CPL 6.3-4

Programs, modules and interfaces

- "Programs" as collections of definitions (with an entry point)
- Namespaces and packages: collecting related components together, using "dot" syntax to structure names; importing namespaces to allow local usage
- The idea of abstract data types: a type with associated operations, with hidden implementation
- Modules (e.g. Scala's objects) and interfaces (e.g. Scala's traits)
- What it means for a module to "implement" an interface
- Reading: CPL 9, PFPL2 42.1-2, 44.1

Objects and classes

- Objects and how they differ from records or modules: encapsulation of local state; self-reference
- Classes and how they differ from interfaces; abstract classes; dynamic dispatch
- Instantiating classes to obtain objects
- Inheritance of functionality between objects or classes; multiple inheritance and its problems
- Run-time type tests and coercions (isInstanceOf, asInstanceOf)
- Reading: CPL 10, 12.5, 13.1-2

Object-oriented functional programming

- Advanced OOP concepts:
  - inner classes, nested classes, anonymous classes/objects
  - Generics: Parameterized types and parametric polymorphism; interaction with subtyping; type bounds
  - Traits as mixins: implementing multiple traits providing orthogonal functionality; comparison with multiple inheritance
- Function types as interfaces
- List comprehensions and `map`, `flatMap` and `filter` functions
- Reading: Odersky and Rompf, Unifying Functional and Object-Oriented Programming with Scala, CACM, Vol. 57 No. 4, Pages 76-86, April 2014

Imperative programming

- L_{While}: a language with statements, variables, assignment, conditionals and loops
- Interpreting L_{While} using *state* or *store*
- Operational semantics of L_{While}
- **[non-examinable]** Structured vs unstructured programming
- **[non-examinable]** Other control flow constructs: goto, switch, break/continue
- Reading: CPL 4.4, 5.1-2, 8.1

Small-step semantics and type safety

- Small-step evaluation relation $e \mapsto e'$, and advantages over big-step semantics for discussing type safety
- Induction on derivations
- Type soundness: decoposition into *preservation* and *progress* lemmas
- Representative cases for L_{If}
- **[non-examinable]** Type soundness for L_{Rec}
- Reading: CPL 6.1-2, PFPL2 5.1-2, 2.4, 7.2, 6.1-2

References and resource management

- Reconciling references and mutability with a "functional" language like L_{Rec}
- Semantics and typing for references
- Potential interactions with subtyping; problem with reference / array types being covariant in e.g. Java
- **[non-examinable]** How references + polymorphism can violate type soundness
- Resources and allocation/deallocation
- Reading: PFPL2 35.1-3, CPL 5.4.5, 13.3

Evaluation strategies

- Evaluation order; varying small-step "administrative" rules to get left-to-right, right-to-left or unspecified operand evaluation order
- Evaluation strategies for function arguments (or more generally for expressions bound to variables):
  - Call-by-value / eager
  - Call-by-name
  - Call-by-need / lazy evaluation
- Interactions between evaluation strategies and side-effects
- Lazy data structures and pure functional programming (cf. Haskell)
- Reading: PFPL2 36.1, CPL 7.3, 8.4

Exceptions and continuations

- Exceptions, illustrated in Java and Scala (throw, try...catch...finally)
- Exceptions more formally: typing and small-step evaluation rules
- Tail recursion
- **[non-examinable]** Continuations
- Reading: CPL 8.2-3, PFPL2 29.1-3, PFPL2 30.1-2

Reading summary

- The following sections of CPL are recommended to provide high-level explanation and background:
  1, 4.1-2, 4.4, 5.4, 6.1-5, 7.1, 7.3, 8.1-4, 9, 10, 12.5, 13.1-3
- The following sections of PFPL2 are recommended to complement the formal content of the course:
  1, 2, 3.1-2, 4.1-2, 5.1-2, 6.1-2, 7.1-2, 8, 19.1-2, 10.1-2, 11.1-3, 16.1, 24.1-3, 35.1-3, 36.1, 42.1-2, 44.1
- (warning: chapter references for 1st edition differ!)
- In general, exam questions should be answerable using ideas introduced/explained in lectures or tutorials
- (please ask, if something mentioned in lecture slides is unclear and not explained in associated readings)

Exam Information

Exam format

- Written exam, 2 hours
- Three (multi-part) questions
- Answer Question 1 + EITHER Question 2 or 3
- Closed-book (no notes, etc.), but...
- Exam will **not** be about memorizing inference rules — any rules needed to construct derivations will be provided in a supplement
- Check University exam schedule!
  - Exam in December $\iff$ you are a visiting student AND only here for semester 1
  - Exam in April/May $\iff$ you are here for full academic year

Expectations

- Several typical kinds of questions...
- Show how to use / apply some technical content of the course (typing rules, evaluation,) — possibly in a slightly different setting than in lectures/assignments
- Define concepts; explain differences/strengths/weaknesses of differerent ideas in PL design
- Show how to extrapolate or extend concepts or technical ideas covered in lectures (possibly in ways covered in more detail in reading or tutorials but not in lectures)
- Explain and perform simple examples of inductive proofs (no more complex than those covered in lectures)

Sample exam

- A sample exam is available now on course web page
- Format: same as real exam
- Questions have not gone through same process, so:
  - There may be errors/typos (hopefully not on real exam)
  - The difficulty level may not be calibrated to the real exam (though I have tried to make it comparable)
- In particular: just because a topic is covered/not covered on the sample exam does NOT tell you it will be / will not be covered on the real exam!
- There will be a **exam review session** on Thursday, November 30 at 3:10pm (usual lecture time/place, 7 Bristo Square LT2)
- Bring questions!

Conclusions

What **didn't** we cover?

- Lots! (course is already dense as it is)
- Scala: implicits, richer pattern matching, concurrency, ...
- More generally:
  - language-support for concurrent programming (`synchronized`, threads, locks, etc.)
  - language support for other computational models (databases, parallel CPU, GPU, etc.)
  - Haskell-style type classes/overloading
  - Logic programming
  - Program verification / theorem proving
  - Analysis and optimisation
  - Implementation and compilation of modern languages
  - Virtual machines

Other relevant courses

- There is a lot more to Programming Languages than we can cover in just one course...
- The following UG4 courses cover more advanced topics related to programming languages:
  - **Advances in Programming Languages**
  - **Types and Semantics for Programming Languages**
  - **Secure Programming**
  - **Parallel Programming Languages and Systems**
  - **Compiler Optimisation**
  - **Formal Verification**
- Many potential supervisors for PL-related UG4, MSc, PhD projects in Informatics — ask if interested!

Other programming languages resources

- *Scottish Programming Languages Seminar*, http://www.dcs.gla.ac.uk/research/spls/
- EdLambda, Edinburgh's mostly functional programming meetup, http://www.edlambda.co.uk
- Informatics *PL Interest Group*, http://wcms.inf.ed.ac.uk/lfcs/research/groups-and-projects/pl/programming-languages-interest-group
- Major conferences: ICFP, POPL, PLDI, OOPSLA, ESOP, CC
- Major journals: ACM TOPLAS, Journal of Functional Programming

A final word

- This has been the third time of teaching this course *Elements of Programming Languages*
- I hope you've enjoyed the course! I did, though there are still some things that probably need work...
- Please do **provide feedback** on the course (both what worked and what didn't)
  - Course surveys **open now, until 10 December**
  - Thanks in advance on behalf of future EPL students!