Programming Language Theory

CSCI-740 Section 01
Prof. Matthew Fluet

MoWeFr 1:25pm – 2:15pm
GAN-2050 / Synchronous Zoom

COVID-19 Reminders:
- Wear a mask (covering nose and mouth)
- Choose an available seat (spread out, obey “do not sit” designations)
  - Seat chosen today will be assigned to you for remainder of semester
- Complete location check-in (QR code)
- No food or drink during class
Programming Language Theory

Course Overview
Today

- Introductions and course mechanics
- Course motivation and goals
- Course overview
- Course pitfalls
Course Description

This course is an introduction to the formal study of programming languages, demonstrating important intellectual tools for the precise description of programming languages and investigating the essential features of programming languages using these tools. Topics include: dynamic semantics (such as operational semantics); static semantics (such as type systems); proofs by induction on structures and derivations; formal treatment of essential programming-language features (such as assignment, scope, functions, objects, and threads). Both written and programming assignments will be required.
Course Goals

The precise description of the semantics of programming languages is required to thoroughly understand the meaning of computer programs. This course studies the formal semantics of programming languages. Students will learn about important intellectual tools such as operational semantics and type systems and will investigate essential features of programming languages using these tools. While the focus is on formal models of small languages, the applicability of these formal models to “real” programming languages will be demonstrated. Students will gain an appreciation of the design decisions (and design mistakes) in extant programming languages and will be prepared to study the programming-languages research literature.

This course does not cover tools and techniques for describing the concrete syntax of programming languages (e.g., scanners and parsers); such topics are covered in CSCI-741 Compiler Construction.
Introductions

Who am I?

▶ Matthew Fluet
▶ Hooked by sophomore PL course
▶ Studied PL in graduate school and beyond
  ▶ theory, implementation, design
  ▶ MLton (a Standard ML compiler)
  ▶ Type- and Control-Flow Analysis (a program analysis refined by types)
  ▶ Manticore (a heterogeneous parallel functional language)
  ▶ Transactional Events (a novel concurrency abstraction)
  ▶ Delta ML (a language for self-adjusting computation)
  ▶ Cyclone (a safe dialect of C w/ region-based memory management)
Introductions

Who am I?

► Matthew Fluet

► Hooked by sophomore PL course

► Studied PL in graduate school and beyond
  ▶ theory, implementation, design
  ▶ MLton (a Standard ML compiler)
    ▶ a whole-program optimizing Standard ML compiler; actively used in both industry and academia
    ▶ ongoing: whole-program compilation of next-gen language features; automatically managing spatial and temporal locality in a high-level garbage-collected parallel functional PL

► Type- and Control-Flow Analysis (a program analysis refined by types)
► Manticore (a heterogeneous parallel functional language)
► Transactional Events (a novel concurrency abstraction)
► Delta ML (a language for self-adjusting computation)
► Cyclone (a safe dialect of C w/ region-based memory management)
Introductions

Who am I?

- Matthew Fluet
- Hooked by sophomore PL course
- Studied PL in graduate school and beyond
  - theory, implementation, design
  - MLton (a Standard ML compiler)
  - Type- and Control-Flow Analysis (a program analysis refined by types)
    - a program analysis refined by types:
      - control-flow (for first-class functions) + type-flow (for polymorphic types)
    - insight: type- and control-flow are mutually beneficial and decidable
    - ongoing: applications; extend to richer type systems
  - Manticore (a heterogeneous parallel functional language)
  - Transactional Events (a novel concurrency abstraction)
  - Delta ML (a language for self-adjusting computation)
  - Cyclone (a safe dialect of C w/ region-based memory management)
Introductions

Who am I?

► Matthew Fluet
► Hooked by sophomore PL course
► Studied PL in graduate school and beyond
  ► theory, implementation, design
  ► MLton (a Standard ML compiler)
  ► Type- and Control-Flow Analysis (a program analysis refined by types)
  ► Manticore (a heterogeneous parallel functional language)
    ► an effort to design and implement a parallel functional PL supporting heterogeneous parallelism (parallelism at multiple levels)
    ► results: nested schedulers with composable cancellation; novel implicitly-parallel constructs; lazy-tree splitting; data-only flattening; GC for multicore NUMA; partial-abort STM
  ► Transactional Events (a novel concurrency abstraction)
  ► Delta ML (a language for self-adjusting computation)
  ► Cyclone (a safe dialect of C w/ region-based memory management)
Introductions

Who am I?

- Matthew Fluet
- Hooked by sophomore PL course
- Studied PL in graduate school and beyond
  - theory, implementation, design
  - MLton (a Standard ML compiler)
  - Type- and Control-Flow Analysis (a program analysis refined by types)
  - Manticore (a heterogeneous parallel functional language)
  - Transactional Events (a novel concurrency abstraction)
    - a novel concurrency abstraction:
      - first-class synchronous message-passing events + atomic transactions
    - insight: atomicity enhances the expressive power of message-passing
- Delta ML (a language for self-adjusting computation)
- Cyclone (a safe dialect of C w/ region-based memory management)
Introductions

Who are you?

- Why are you taking Programming Language Theory?
- What topics in Programming Language Theory are you especially interested in learning about?
- What concerns do you have about the course?

Homework #00

- A brief, but extremely useful survey, on myCourses
- Things like your background, concerns about the course, etc.
- (Also helps me learn names)
Course Administration

Instructor: Matthew Fluet

▶ E-mail: mtf@cs.rit.edu

▶ Virtual Office Hours: https://rit.zoom.us/my/mtfvcs
  Mon. 3:00pm–4:00pm
  Wed. 10:00am–11:00am
  Wed. 3:00pm–4:00pm
  Fri. 10:00am–11:00am

Website

▶ https://www.cs.rit.edu/~mtf/teaching/20201/plt
  ▶ syllabus, schedule, notes, assignment writeups, resources

▶ https://mycourses.rit.edu
  ▶ Zoom recordings, assignment submissions, discussions, surveys, grades
Assignments, Exams, & Grades

► 5% — Participation
► 50% — Homework Assignments (≈ 7)
► 10% — Technical Perspective
► 10% — Mid-term Exam 1 (Sep. 21 (Mon.); take home/online)
► 10% — Mid-term Exam 2 (Oct. 21 (Wed.); take home/online)
► 15% — Final exam (Dec. ??; take home/online)

More details in syllabus and on course website.
Attendance & Participation

Attendance (in person or via Zoom) is strongly encouraged

- Will post slides to website (after lectures)
- Take notes
  - (not everything from a lecture is in the slides and/or texts)
- Missing the first 10mins is more costly than missing the last 10mins

Participation means being an engaged student

- Asking and answering questions (in person, via Zoom, on Discussions)
  - (not simply attending class)
- Use myCourses Discussion threads to clarify lecture material
  - (understanding that Zoom lectures are sub-optimal for asking questions)
- Let me know if pace is too fast or too slow
- When I enter your grade, will I know who you are?
Flex Options

Course is being offered with both the Online Flex and Full Flex options.

- choose to complete all course requirements online either for the entire semester (Online Flex) or as needed throughout the semester (Full Flex)
- expected to attend class meetings synchronously using Zoom and participate via Zoom and on Discussions (perhaps with a slight emphasis on Discussions)
- contact me (at any time during the semester) if you wish to select one of these options
Homework

Approx. 7 homework assignments

- written ("pencil and paper") components
  - electronic preparation suggested, \texttt{LaTeX} recommended
  - handwritten accepted, scan to PDF for submission (Microsoft Lens or Adobe Scan)

- programming components
  - Standard ML required

- worth the most, where you will learn the most

- start early

Submitted work must be your own

- Written and/or programmed by you alone
- Feel free to discuss homeworks (but do not share solutions)
- Do not search for assignments and/or solutions from other institutions
Technical Perspective

*Communications of the ACM* republishes CS research articles, each introduced by a “Technical Perspective”:

- written by an expert for non-expert computer scientists
- explains the importance of the research
- explains the specific contributions of the paper
- helps to place the paper within a larger research context

Prepare a “Technical Perspective” for a recent PL research paper:

- choose a research paper
- read and understand the paper
  - will require reading additional background material
- write a 1 – 3 page “Technical Perspective”

More details on course website and after Mid-term Exam 2.
Academic Integrity & Late Policy

Academic Integrity

▶ Read course policy (and linked policies)

Late Policy

▶ Assignments generally due on Mon, Wed, or Fri at 5pm
▶ 3 “extension tokens” (for ordinary circumstances)
  ▶ grants a 24-hour extension on a single homework assignment (excludes technical perspective)
  ▶ automatically applied to “Late Submissions” on myCourses (submitted after “Due Date” but before “End Date”)
  ▶ only one extension per assignment
  ▶ won’t answer questions about assignment after “Due Date”
▶ for extraordinary circumstances (incl. COVID-19), contact an appropriate administrative staff member; will make special arrangements suited to the situation
Textbook(s)

Suggested:

- *Types and Programming Languages*, Benjamin Pierce
- *Practical Foundations of Programming Languages*, Robert Harper

Additional:

- *Advanced Types and Programming Languages*, Benjamin Pierce (ed.)
- *The Formal Semantics of Programming Languages*, Glynn Winskell

- Won’t follow any very closely
- Recommended readings on course website
- Good resources to “go further”
Why study programming languages?

A question:
- What is the best programming language?
Why study programming languages?

A question:
- What is the best programming language?
- What is the best kind of car?

An answer:
- It depends on what you are doing.
- Programming languages have many goals, including making it easy in your domain to:
  - Write correct code
  - Write fast code
  - Write code fast
  - Write large projects
  - Interoperate . . .
Why study programming languages?

A question:
▶ What is the best programming language?
  ▶ What is the best kind of car?

An answer:
▶ It depends on what you are doing.
▶ Programming languages have many goals, including making it easy in your domain to:
  ▶ Write correct code
  ▶ Write fast code
  ▶ Write code fast
  ▶ Write large projects
  ▶ Interoperate
  ▶ ...
Why study programming languages?

Another question:

- Aren’t all programming languages the same?
Why study programming languages?

Another question:

▶ Aren’t all programming languages the same?
  ▶ Aren’t all cars the same?
Why study programming languages?

Another question:

- Aren’t all programming languages the same?
  - Aren’t all cars the same?

Three answers:

- Yes: Any input-output behavior that you can program in language X you can also program in language Y
  - Java, Standard ML, and a language with one while loop and three infinite-precision integers are all “equal”
  - Called the “Turing tarpit”

- Yes: Certain fundamentals appear in most languages
  - (variables, abstractions, recursive definitions, …)
  - Travel to appreciate where you’re from

- No: Real differences at formal and informal levels
  - A program in language X might be very different from the “same” program in language Y
Why study programming languages?

Yet another question

► Will I ever need to design a programming language?

Yes: If you design an extensible software system or a non-trivial API, then you are designing a (small?) programming language

Examples: VBScript, JavaScript, QuakeC, Renderman, bash, AppleScript, emacs, Eclipse, AutoCAD, . . .

"Greenspun's Tenth Rule of Programming: any sufficiently complicated C or Fortran program contains an ad hoc informally-specified bug-ridden slow implementation of half of Common Lisp."

— Philip Greenspun
Why study programming languages?

Yet another question

▶ Will I ever need to design a programming language?
  ▶ Will I ever need to design a car?

— Philip Greenspun

Greenspun’s Tenth Rule of Programming: any sufficiently complicated C or Fortran program contains an ad hoc informally-specified bug-ridden slow implementation of half of Common Lisp.

Matthew Fluet
Programming Language Theory
Lecture 01
Why study programming languages?

Yet another question

► Will I ever need to design a programming language?
  ► Will I ever need to design a car?

► Yes: If you design an extensible software system or a non-trivial API, then you are designing a (small?) programming language
  ► Examples: VBScript, JavaScript, QuakeC, Renderman, bash, AppleScript, emacs, Eclipse, AutoCAD, . . .
  ► “Greenspun’s Tenth Rule of Programming: any sufficiently complicated C or Fortran program contains an ad hoc informally-specified bug-ridden slow implementation of half of Common Lisp.”

—— Philip Greenspun
How to study programming languages?

The wrong question (for this class):

- What programming language should we study?

Non-answers (for this class):

- What libraries are available?
- What does management/client want?
- What is the de facto industry standard?
- ...
How will we study programming languages?

- We will investigate *universal foundations* and *essential concepts* in programming languages
- We will learn intellectual tools for describing programming languages and programs

- Define a language *precisely*:
  - English is a poor *metalanguage*.
- Focus on *semantics*:
  - What do programs mean (do/compute/produce/represent)?
  - Other aspects of meaning: equivalence, termination, determinism, type, . . .
Why study programming languages this way?

Novices write programs that “work as expected,” so why be rigorous/precise/pedantic?

- The world runs on software
  - web-servers and nuclear reactors don’t “seem to work”
- You buy/use language implementations
  - what do they do?
- Software is buggy
  - semantics assigns blame
- Real languages have many features
  - building them from well-understood foundations is good engineering
- Never say “nobody would write that”
  - surprising interactions
Why study programming languages this way?

Building a rigorous and precise model is a hallmark of quality work.

The value of a model is in its:

▷ fidelity
▷ convenience for establishing (and proving) properties
▷ revealing alternatives and design decisions
▷ ability to communicate ideas concisely

Why we mostly do it for programming languages:

▷ Elegant things we all use
▷ Remarkably complicated (need rigor)

But this “theory” makes you a better computer scientist

▷ focus on the model-building (applicable everywhere), not just the PL features
Course goals

1. Learn intellectual tools for describing programming languages
   - operational semantics
   - type systems

2. Investigate essential concepts in programming languages
   - control flow, including iteration and recursion
   - functions, objects, continuations, threads
   - assignment and mutation
   - types
   - ...

3. Write programs to “connect theory with code”
   - Standard ML

4. Sketch applicability to “real” languages

5. Provide background for current PL research
   - (less important for most of you)
Course non-goals

► Learn to specify and recognize lexical and syntactic structure
  ► (i.e., no scanners or parsers)
► Learn specific programming languages
  ► (but some ML)
How will we study programming languages?

- Define really small languages
  - Usually Turing complete
  - Always unsuitable for real programming
- Study them rigorously via *operational semantics* and *type systems*
- Extend them to realistic languages less rigorously
- Digress for cool results (this is fun!?!)

- Do programming assignments in Standard ML...
Standard ML

- Standard ML is an awesome, high-level language
- We will use a core subset of it that is well-suited for manipulating recursive data structures (like programs!)
- You mostly have to learn it outside of class (start today!)
  - Resources on course website
  - Read Robert Harper’s Programming in Standard ML
  - Feel free to ask questions
- Knowing ML makes you a better programmer
  - (as does knowing Logo, Prolog, Smalltalk, etc.)
Pitfalls

How to hate this course and get the wrong idea:

▶ Forget that we made simple models to focus on essentials
▶ Don’t quite get inductive definitions and proofs
▶ Don’t try other ways to model/prove the idea
  ▶ You’ll probably be wrong
  ▶ And therefore you’ll learn more
▶ Think PL people focus on only obvious facts
  ▶ (need to start there)
Final Metacomment

Acknowledging others is crucial...

This course will draw heavily on:

- similar courses elsewhere:
  - University of Washington (Dan Grossman)
  - Cornell University (Andrew Myers, Dexter Kozen, Nate Foster)
  - Indiana University (Amal Ahmed)
  - Harvard University (Greg Morrisett)
  - ...

- texts:
  - Pierce (*TAPL*)
  - Winskel (*FSPL*)
  - Harper (*PFPL*)
  - ...

This is a course, not my original content.