EECS 321
Programming Languages

Spring 2019

Instructor: **Vincent St-Amour**

Course Details

`http://www.eecs.northwestern.edu/~stamourv/teaching/321-S19`

(or search for “Vincent St-Amour” and follow the links)

- Slides will be posted there
- Office hours and logistic info
- Link to Piazza
- Everything, really

Course Format

- 9 homework assignments
    - Grades: check+ (A), check (B), check- (C), 0 (F)
    - Assignments due Fridays at 6pm
    - Grades out on Mondays (or we give you a heads up)
    - Individual submissions
- No exams
- A lot of programming, all in Racket*
    - We assume you either know Racket
    - or can pick it up on your own
- Loosely following PLs: Application and Interpretation (PLAI)
    - First edition
    - Link on course web page

Resubmission Policy

- I care that you learn the material, sooner or later
  - So you should get credit for it, even if it takes you more time
  - But I still want deadlines to keep you on top of things
- You can resubmit for up to two weeks after an assignment's deadline
  - That's two more chances to get feedback (and a grade)
- Resubmission grades capped halfway between "check" and "check+"
  - Even if you got a "check", you still have things to learn
  - So you should get credit if you do
- I'm giving you a lot of rope; use it carefully
  - Don't fall behind
- Details on the course web page

Academic Integrity

- Collaboration good, plagiarism bad
  - You need to understand the difference
- The work you submit must be your own
- Don't even ***look*** at other solutions!
  - Not your colleagues'
  - Not online
- We check
- We report anything suspicious to the dean

Classroom Etiquette

- Learning this (or any) material requires focus and concentration
  - Let's ensure our classroom environment is conducive to that
- Laptops
  - Laptops are fine; following along with the slides is great
  - Some laptop activities are distracting, though
    - If you're planning to do non-course-related stuff, sit in the back
    - So you don't distract your colleagues who are paying attention
- Talking
  - Asking a quick question to your neighbor is fine
    - But ask me instead, so everyone benefits from the answer
  - Continuous talking is extremely rude
    - Distracting for your colleagues around you, and for me too
    - If you want to chat, go outside

Course Staff and Office Hours

Instructor: Vincent St-Amour
Wednesdays 1-2 or by appointment, Mudd 3215

Peer mentors: Chloe Brown, Hakan Dingenc, Kate Hayner-Slattery, Jeremy Kaish, Louisa Lee, Patrick Sachaj
See web site

TA: Spencer Florence
(Will help peer mentors at busy times.)

Key Ideas of this Class

Programs are data

- ... which other programs can operate on
    - to run then (interpreters)
    - to transform them (refactoring tools)
    - to check properties about them (type checkers)
- ... which other programs can generate
    - to make them more efficient (compilers)
    - to automate some aspects of programming (code generators)
    - to generate infinite test cases (generative testing)
- Comes up surprisingly often in practice!
    - ... if you know how to look!

Key Ideas of this Class

Meta-language vs object-language

- Meta-language programs operate on programs in the object-language

- Our meta-language will be a variant of Racket: `#lang plai`
    - Very well suited as a meta-language

- Our object-languages will be many, small, and simple
    - Designed to illustrate specific concepts

Key Ideas of this Class

Different languages share common concepts

- Found in the vast majority of languages:
  - Operations on basic data (e.g., arithmetic)
  - Variables and scope
  - Functions
  - State
  - etc.
- Learn those well, and learning languages is easy!
  - New faces on familiar ideas
  - Languages go and come, but λ abides
- Differences between languages as variations on such concepts
  - (Most of) the rest is (mostly) cosmetic
  - Then you can focus on the differences that **do** matter

This Class's Approach

Learn by building

- Both in lecture and in homeworks

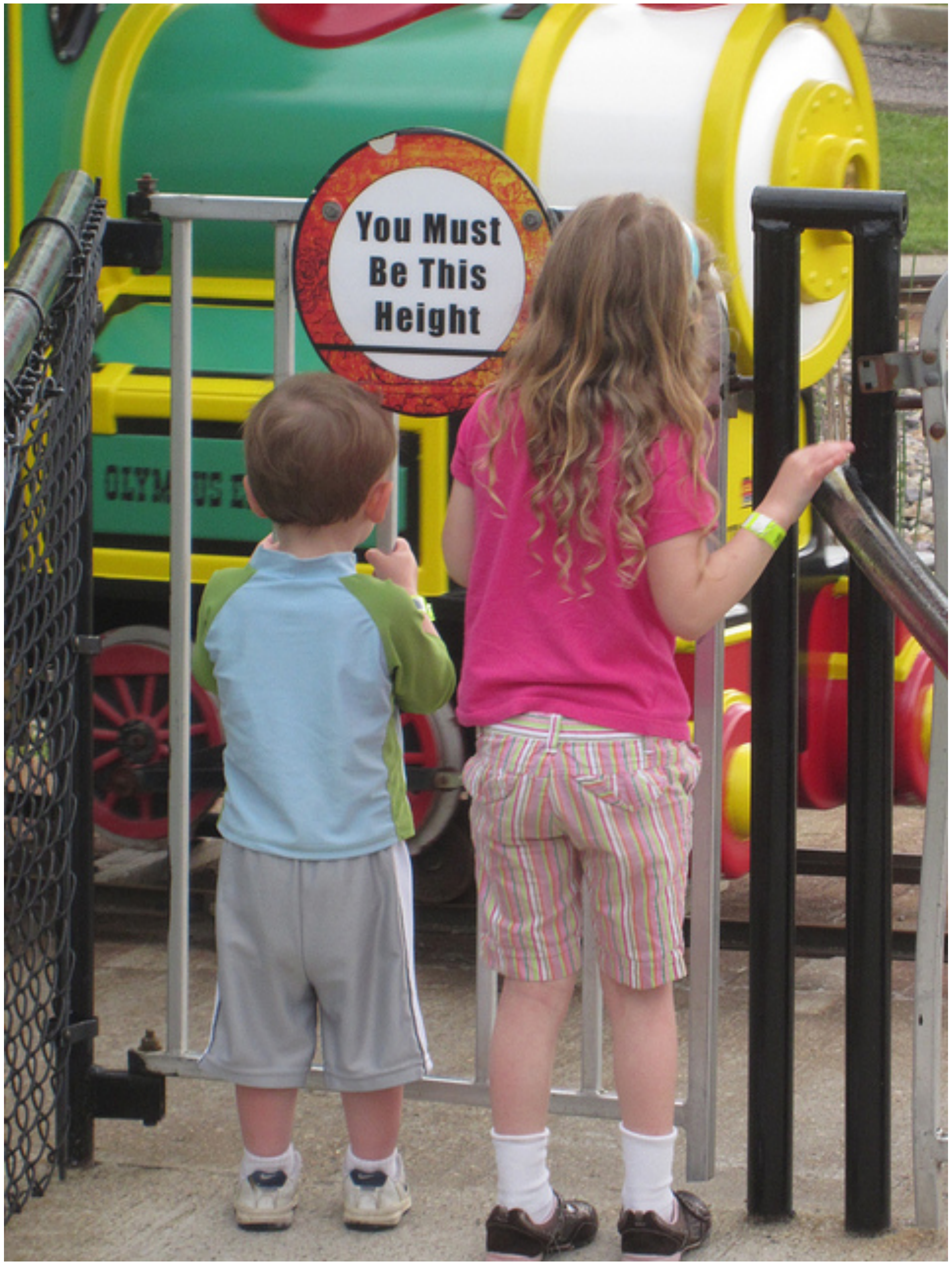
- We will build interpreters
  - Interpreter = (meta-language) program that executes (object-language) programs
  - New concept → new object-language → new interpreter
  - See more than one way to implement most concepts

- ... and also a few other programs that operate on programs
  - Parsers
  - Program generators
  - Compilers
  - Type checkers

Topic Outline

- Variables and binding (substitution and deferred substitution)
- (Higher-order) functions
- Parsing (a little)
- Random testing
- Recursion
- State
- Control
- Garbage collection
- Type checking and type inference

Homework #1



On the course web page
Due on Friday at 6pm

To test your prerequisites
Should be very easy
If not, you may not be ready

Tree traversals and manipulations
will be our bread and butter
So you need to master them!

Future homeworks:
Also due on Fridays at 6pm

Tutorial Session

- If you feel rusty on Racket-related concepts
- Tonight 6pm, Tech M152

Let's dive in!

The Most Common Kinds of Program Manipulators

An ***interpreter*** takes a program and produces a result

- Python
- **`bash`**
- Racket
- x86 processor
- Desktop calculator
- Algebra student

Good for understanding program behavior, easy to implement (our focus)

A ***compiler*** takes a program and produces a program

- **`gcc`**
- **`javac`**
- Racket
- x86 processor

Good for speed, more complex (take 322)

So, what's a ***program***?

A Grammar for Algebra Programs

A grammar of Algebra in ***BNF*** (Backus-Naur Form):

⟨prog⟩	::=	⟨defn⟩* ⟨expr⟩
⟨defn⟩	::=	⟨id⟩(⟨id⟩) = ⟨expr⟩
⟨expr⟩	::=	(⟨expr⟩ + ⟨expr⟩)
	|	(⟨expr⟩ - ⟨expr⟩)
	|	⟨id⟩(⟨expr⟩)
	|	⟨id⟩
	|	⟨num⟩
⟨id⟩	::=	a variable name: **f**, **x**, **y**, **z**, ...
⟨num⟩	::=	a number: 1, 42, 17, ...

Each ***meta-variable***, such as ⟨prog⟩, defines a set

Using a BNF Grammar

$\langle$id$\rangle$ ::= a variable name: **f**, **x**, **y**, **z**, ...

$\langle$num$\rangle$ ::= a number: 1, 42, 17, ...

The set $\langle$id$\rangle$ is the set of all variable names

The set $\langle$num$\rangle$ is the set of all numbers

To make an example member of $\langle$num$\rangle$, simply pick an element from the set

$$2 \in \langle \text{num} \rangle$$

$$298 \in \langle \text{num} \rangle$$

Using a BNF Grammar

$$\begin{array}{lcl} \langle \text{expr} \rangle & ::= & (\langle \text{expr} \rangle + \langle \text{expr} \rangle) \\ & | & (\langle \text{expr} \rangle - \langle \text{expr} \rangle) \\ & | & \langle \text{id} \rangle(\langle \text{expr} \rangle) \\ & | & \langle \text{id} \rangle \\ & | & \langle \text{num} \rangle \end{array}$$

The set ⟨expr⟩ is defined in terms of other sets

We'll have to do this in steps

Using a BNF Grammar

⟨expr⟩	::=	(⟨expr⟩ + ⟨expr⟩)
	|	(⟨expr⟩ - ⟨expr⟩)
	|	⟨id⟩(⟨expr⟩)
	|	⟨id⟩
	|	⟨num⟩

To make an example ⟨expr⟩:

- choose one case in the grammar
- pick an example for each meta-variable
- combine the examples with literal text

Using a BNF Grammar

$$
\begin{array}{lcl}
\langle \text{expr} \rangle & ::= & (\langle \text{expr} \rangle + \langle \text{expr} \rangle) \\
& | & (\langle \text{expr} \rangle - \langle \text{expr} \rangle) \\
& | & \langle \text{id} \rangle(\langle \text{expr} \rangle) \\
& | & \langle \text{id} \rangle \\
& | & \langle \text{num} \rangle
\end{array}
$$

To make an example $\langle \text{expr} \rangle$:

- choose one case in the grammar
- pick an example for each meta-variable

$$7 \in \langle \text{num} \rangle$$

- combine the examples with literal text

$$7 \in \langle \text{expr} \rangle$$

Using a BNF Grammar

$$
\begin{array}{lcl}
\langle\text{expr}\rangle & ::= & (\langle\text{expr}\rangle + \langle\text{expr}\rangle) \\
 & | & (\langle\text{expr}\rangle - \langle\text{expr}\rangle) \\
 & | & \langle\text{id}\rangle(\langle\text{expr}\rangle) \quad \Leftarrow \\
 & | & \langle\text{id}\rangle \\
 & | & \langle\text{num}\rangle
\end{array}
$$

To make an example $\langle\text{expr}\rangle$:

- choose one case in the grammar
- pick an example for each meta-variable

$$\mathbf{f} \in \langle\text{id}\rangle \qquad\qquad 7 \in \langle\text{expr}\rangle$$

- combine the examples with literal text

$$\mathbf{f}(7) \in \langle\text{expr}\rangle$$

Using a BNF Grammar

$$\begin{array}{lcl} \langle \text{expr} \rangle & ::= & (\langle \text{expr} \rangle + \langle \text{expr} \rangle) \\ & | & (\langle \text{expr} \rangle - \langle \text{expr} \rangle) \\ & | & \langle \text{id} \rangle(\langle \text{expr} \rangle) \\ & | & \langle \text{id} \rangle \\ & | & \langle \text{num} \rangle \end{array}$$

To make an example ⟨expr⟩:

- choose one case in the grammar
- pick an example for each meta-variable

  f ∈ ⟨id⟩ **f**(7) ∈ ⟨expr⟩

- combine the examples with literal text

  f(**f**(7)) ∈ ⟨expr⟩

Using a BNF Grammar

$$\langle\text{prog}\rangle \quad ::= \quad \langle\text{defn}\rangle^{*}\ \langle\text{expr}\rangle$$

$$\langle\text{defn}\rangle \quad ::= \quad \langle\text{id}\rangle(\langle\text{id}\rangle) = \langle\text{expr}\rangle$$

$$\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1) \in \langle\text{defn}\rangle$$

To make a ⟨prog⟩ pick some number of ⟨defn⟩s

$$(\mathbf{x} + \mathbf{y}) \in \langle\text{prog}\rangle$$

$$\begin{array}{l} \mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1) \\ \mathbf{g}(\mathbf{y}) = \mathbf{f}((\mathbf{y} - 2)) \quad \in \langle\text{prog}\rangle \\ \mathbf{g}(7) \end{array}$$

So, what's a language, then?

A ***programming language*** is defined by

- a grammar that describes what programs are possible
- rules for evaluating any such program to produce a result

For example, algebra evaluation is defined in terms of evaluation steps:

(2 + (7 - 4)) $\longrightarrow$ (2 + 3) $\longrightarrow$ 5

So, what's a language, then?

A ***programming language*** is defined by

- a grammar that describes what programs are possible
- rules for evaluating any such program to produce a result

For example, algebra evaluation is defined in terms of evaluation steps:

$$\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1)$$

$$\mathbf{f}(10) \quad \rightarrow \quad (10 + 1) \quad \rightarrow \quad 11$$

Evaluation

- Evaluation ($\rightarrow$) is defined by a set of pattern-matching rules:

$$(2 + (7 - 4)) \quad \rightarrow \quad (2 + 3)$$

due to the rule

$$\ldots (7 - 4) \ldots \quad \rightarrow \quad \ldots 3 \ldots$$

Evaluation

- Evaluation ($\rightarrow$) is defined by a set of pattern-matching rules:

$$\mathbf{f}(\mathbf{x}) = (\mathbf{x} + 1)$$

$$\mathbf{f}(10) \quad \rightarrow \quad (10 + 1)$$

due to the rule

$$\textbf{...}\ \langle \mathrm{id} \rangle_{\mathbf{1}}(\langle \mathrm{id} \rangle_{\mathbf{2}}) = \langle \mathrm{expr} \rangle_{\mathbf{1}}\ \textbf{...}$$

$$\textbf{...}\ \langle \mathrm{id} \rangle_{\mathbf{1}}(\langle \mathrm{expr} \rangle_{\mathbf{2}})\ \textbf{...} \quad \rightarrow \quad \textbf{...}\ \langle \mathrm{expr} \rangle_{\mathbf{3}}\ \textbf{...}$$

where $\langle \mathrm{expr} \rangle_{\mathbf{3}}$ is $\langle \mathrm{expr} \rangle_{\mathbf{1}}$ with $\langle \mathrm{id} \rangle_{\mathbf{2}}$ replaced by $\langle \mathrm{expr} \rangle_{\mathbf{2}}$

Rules for Evaluation

- **Rule 1:** one pattern

$$\ldots \langle id \rangle_1 (\langle id \rangle_2) = \langle expr \rangle_1 \ldots$$

$$\ldots \langle id \rangle_1 (\langle expr \rangle_2) \ldots \quad \rightarrow \quad \ldots \langle expr \rangle_3 \ldots$$

where $\langle expr \rangle_3$ is $\langle expr \rangle_1$ with $\langle id \rangle_2$ replaced by $\langle expr \rangle_2$

- **Rules 2 -** ∞: special cases

... (0 + 0) ... → ... 0 ...	... (0 - 0) ... → ... 0 ...
... (1 + 0) ... → ... 1 ...	... (1 - 0) ... → ... 1 ...
... (2 + 0) ... → ... 2 ...	... (2 - 0) ... → ... 2 ...
etc.	*etc.*

When the interpreter is a program instead of an Algebra student, the rules look a little different

Action Items

- Sign up for Piazza
- Brush up your Racket
- Read the docs for the PLAI language
  (comes with Racket)
  http://docs.racket-lang.org/plai/plai-scheme.html
- Do Homework 1