CPSC 311: Definition of Programming Languages

2015 Winter Term 1

Joshua Dunfield

University of British Columbia

2015–09–09: Lecture 1

`www.ugrad.cs.ubc.ca/~cs311`

Today

- Who am I?
- Hello, what is this course about?
- Defining “Definition of Programming Languages”
- Logistics (attendance, marks, etc., etc.)
- Next time. . .

Who am I?

"Joshua"

- When I taught at McGill, some students called me "Sir". That felt strange.
- In Germany, I got mail addressed to "Herr Dr. Dunfield". That felt even stranger.

Who am I?

"Joshua" = Research Associate + Sessional Lecturer

- A postdoc is sort of halfway between graduate student and professor
- A "Research Associate" is. . . halfway between postdoc and professor?
- I've studied programming languages for a long time

Who am I?

"Joshua" = Research Associate + Sessional Lecturer

- A postdoc is sort of halfway between graduate student and professor
- A "Research Associate" is. . . halfway between postdoc and professor?
- I've studied programming languages for a long time
- I've studied Racket for a **much** shorter time. . .

Hello, what is this course about?

- World domination?

World domination: this time for sure!

- 196x: Algol was going to dominate

World domination: this time for sure!

- 196x: Algol was going to dominate
- 1970: PL/I was going to dominate

World domination: this time for sure!

- 196x: Algol was going to dominate
- 1970: PL/I was going to dominate
- 1980: C was going to dominate

World domination: this time for sure!

- 196x: Algol was going to dominate
- 1970: PL/I was going to dominate
- 1980: C was going to dominate
- 1990: C++ was **totally** going to dominate [Oak paper]

World domination: this time for sure!

- 196x: Algol was going to dominate
- 1970: PL/I was going to dominate
- 1980: C was going to dominate
- 1990: C++ was **totally** going to dominate [Oak paper]
- 2000: Java was going to dominate

World domination: this time for sure!

- 196x: Algol was going to dominate
- 1970: PL/I was going to dominate
- 1980: C was going to dominate
- 1990: C++ was **totally** going to dominate [Oak paper]
- 2000: Java was going to dominate

Is there some kind of pattern here?

World domination: this time for sure!

- 196x: Algol was going to dominate
- 1970: PL/I was going to dominate
- 1980: C was going to dominate
- 1990: C++ was **totally** going to dominate [Oak paper]
- 2000: Java was going to dominate

Is there some kind of pattern here?

(We might be learning, finally...)

311: Unimpressed by fads

- 311 is not about learning a little about a lot
  ("trip to the zoo")
- We will focus on learning concepts and methods that should improve PLs in the long run
- Good PL ideas get adopted. . . eventually.
  (Lisp in the '60s $\Rightarrow$ Java in the '90s)
- Some hopeful signs that it's getting faster;
  Rust has stuff invented only 10-15 years ago.

311: Unimpressed by fads

- 311 is not about learning a little about a lot
  ("trip to the zoo")
- We will focus on learning concepts and methods that should improve PLs in the long run
- Good PL ideas get adopted. . . eventually.
  (Lisp in the '60s $\Rightarrow$ Java in the '90s)
- Some hopeful signs that it's getting faster;
  Rust has stuff invented only 10-15 years ago.

This cat is unimpressed by fads.

Course goals

You will learn how to

- **Understand** design choices (scope, evaluation order, types...) and some arguments for (and against) them
- **Understand**, **modify**, and **reason about** definitions of programming languages
- **Implement** interpreters for programming languages

Magic-free zone

- Programming languages aren't magic

Magic-free zone

- Programming languages aren't magic
- But they're still lots of fun!

Magic-free zone

- Programming languages aren't magic
- But they're still lots of fun!
- Programming:
  "I can tell the computer what to do"

Magic-free zone

- Programming languages aren't magic
- But they're still lots of fun!
- Programming:
  "I can tell the computer what to do"
- Programming languages:
  "I can tell the computer **how to understand the instructions**"

Definition of “Programming Languages”

What is a programming language?

- A way to instruct computers

Definition of “Programming Languages”

What is a programming language?

- A way to instruct computers
- A well-defined way to instruct computers

Definition of "Programming Languages"

What is a programming language?

- A way to instruct computers
- A well-defined way to instruct computers
- A well-defined way to instruct computers, using symbols

Definition of "Programming Languages"

What is a programming language?

- A way to instruct computers
- A well-defined way to instruct computers
- A well-defined way to instruct computers, using symbols

Computers compute.
A programming language is a
precise, **symbolic** description of a set of possible computations.

Definition of Programming Languages

A programming language is a
precise, **symbolic** method of describing computations.

Caveats:

Definition of Programming Languages

A programming language is a
precise, **symbolic** method of describing computations.

Caveats:

- "Symbolic": occasional attempts at visual PLs
  (Smalltalk-80? Logo? Prograph)

Definition of Programming Languages

A programming language is a
precise, **symbolic** method of describing computations.

Caveats:

- "Symbolic": occasional attempts at visual PLs (Smalltalk-80? Logo? Prograph)
- "Precise" is often aspirational...

Definition of Programming Languages

A programming language is a
precise, **symbolic** method of describing computations.

Definition of Programming Languages

A programming language is a
precise, **symbolic** method of describing computations.

- **Programmers** need precision so they know what programs are supposed to do.

Definition of Programming Languages

A programming language is a
precise, **symbolic** method of describing computations.

- **Programmers** need precision so they know what programs are supposed to do.
- Language **implementors** need precision so they know how to implement (interpret, compile, translate to another language) a language.

Definition of Programming Languages

A programming language is a
precise, **symbolic** method of describing computations.

- **Programmers** need precision so they know what programs are supposed to do.
- Language **implementors** need precision so they know how to implement (interpret, compile, translate to another language) a language.
- Unfortunately, most PLs are defined using English;
  a few are defined using math/logic.

Definition of Programming Languages

A programming language is a
precise, **symbolic** method of describing computations.

- **Programmers** need precision so they know what programs are supposed to do.
- Language **implementors** need precision so they know how to implement (interpret, compile, translate to another language) a language.
- Unfortunately, most PLs are defined using English;
  a few are defined using math/logic.
- Unclear what **can** be defined,
  and what **should** be defined:
  "The C language does not exist"
  (from *Communications of the ACM*)

Definition of Programming Languages

A programming language is a
precise, **symbolic** description of a set of possible computations.

- A key idea in programming language research:
  There are deep connections between (some) PLs and (some) **logics**.

Definition of Programming Languages

A programming language is a
precise, **symbolic** description of a set of possible computations.

- A key idea in programming language research:
  There are deep connections between (some) PLs and (some) **logics**.
- PL = system of computation
  logic = system of reasoning

Definition of Programming Languages

A programming language is a
precise, **symbolic** description of a set of possible computations.

- A key idea in programming language research:
  There are deep connections between (some) PLs and (some) **logics**.
- PL = system of computation
  logic = system of reasoning
- A proof of "if X, then Y" is like a function of type $X \to Y$.

Definition of Programming Languages

A programming language is a
precise, **symbolic** description of a set of possible computations.

- A key idea in programming language research:
  There are deep connections between (some) PLs and (some) **logics**.
- PL = system of computation
  logic = system of reasoning
- A proof of "if X, then Y" is like a function of type $X \to Y$.
- We'll probably only touch on this in 311.

Three sides of PLs

- 1. **Syntax** describes **which sequences of symbols are reasonable**.
- 2. **Dynamic semantics** describes **how to run programs**.
- 3. **Static semantics** describes **what programs are**.

1. Syntax

- Syntax is (usually) the easiest to define, understand, and process.

1. Syntax

- Syntax is (usually) the easiest to define, understand, and process.
- Racket makes it even easier than usual!

  (By accident: the inventors of Lisp designed a more complex syntax, but the simple syntax had already spread. For once, simplicity won.)

1. Syntax

- Syntax is (usually) the easiest to define, understand, and process.
- Racket makes it even easier than usual!

  (By accident: the inventors of Lisp designed a more complex syntax, but the simple syntax had already spread. For once, simplicity won.)
- We won't spend much time on syntax.

2. Dynamic semantics

Dynamic semantics is about **how** programs behave:

- Dynamic semantics tells you how to "step" a program.
- You can't ride a bus effectively unless you know that buses tend to move forward.

3. Static semantics

Static semantics is about **what** programs are.

- Static semantics tells you how to understand a program **without** stepping it.
- You don't want to experimentally ride every bus until you get where you want to be.
  ("See where it takes you"?!)

Defining dynamic semantics

- **Rules** define how to step a program:

$$\frac{V1 \in \mathbb{Z} \qquad V2 \in \mathbb{Z} \qquad n = V1 + V2}{(+\ V1\ V2) \mapsto n}$$

$$\frac{E1 \mapsto E2}{(V\ E1\ \ldots) \mapsto (V\ E2\ \ldots)}$$

- Reminiscent of the "laws of computation" from *How to Design Programs*: BSL Intermezzo

Defining static semantics

- A [static] **type system** keeps out sort-of-nonsense:

```
(+ "no" 1)
```

Defining static semantics

- A [static] **type system** keeps out sort-of-nonsense:

```
(+ "no" 1)
```

- Like stepping, type systems can be defined by rules.

$$\frac{\texttt{E1} : \text{number} \qquad \ldots \qquad \texttt{En} : \text{number}}{\texttt{(+ E1 ... En)} : \text{number}}$$

Prerequisites

- Official prerequisite: CPSC 210
- At least as helpful: CPSC **110**
  - . . . because in 110, you programmed in Racket.

Prerequisites

- Official prerequisite: CPSC 210
- At least as helpful: CPSC **110**
  - . . . because in 110, you programmed in Racket.
  - If you don't know Racket (Scheme), you'll need to spend extra time on 311, especially in the first few weeks!

Prerequisites

- Official prerequisite: CPSC 210
- At least as helpful: CPSC **110**
  - . . . because in 110, you programmed in Racket.
  - If you don't know Racket (Scheme), you'll need to spend extra time on 311, especially in the first few weeks!
  - If you've forgotten Racket, you'll need to spend some extra time.

Texts

- For the first couple of weeks, and again near the end, we'll **roughly** follow some chapters from Shriram Krishnamurthi's *Programming Languages: Application and Interpretation*.
- For the middle part of the course, we'll use my lecture notes, supplemented with other materials.
- Everything we use will be available for free on the web.

Lectures

- Mix and match:
  slides, DrRacket on my laptop, whiteboard,
  camera projector, . . .
- We will often develop code, rules, or ideas **on the fly**.
- I do **not** grade attendance or participation.
- But you'll do better if you attend and participate, especially since we're not strictly following a textbook.
- My lecture notes will be **intended** to be complete,
  but intent is not magic.

TAs

- Tutorials in X-Wing 015 by your awesome TAs:
  - Mon. 12:00–13:00 Felipe Bañados Schwerter
  - Mon. 15:00–16:00 Louie Dinh
  - Mon. 16:00–17:00 Yan Peng
- TA office hours (probably also in X-Wing)
  to be determined

Piazza

- Discussions on our Piazza site (link on course webpage)
  - I haven't used Piazza before, so bear with me.

Marking

- **Assignments, including project:** 45%
  - Assignments (some in groups)
  - Group project
- **Midterm exam:** 15%
- **Final exam:** 40%
- Midterm/final are "all's well that ends well":
  - If your final exam score is **higher** than your midterm score, the final is "inflated" to 55%.

Marking

- **Assignments, including project:** 45%
  - Assignments (some in groups)
  - Group project
- **Midterm exam:** 15%
- **Final exam:** 40%
- Midterm/final are "all's well that ends well":
  - If your final exam score is **higher** than your midterm score, the final is "inflated" to 55%.
- The instructor reserves the right to modify these weights (but does not anticipate exercising that right).

Assignments

- Partly **programming** (mostly in Racket):
  - implementing **dynamic semantics** by writing interpreters (**stepping** programs according to rules)
  - implementing **static semantics** by writing type checkers, according to rules
- Partly **theory**
  (is theory anything that isn't programming?)

Survey

- Mostly for us to decide how much time to spend on Racket review

Next time...

`www.ugrad.cs.ubc.ca/~cs311`

- Start refreshing your Racket:
  110 material, HtDP, etc. (see website)
- Skim PLAI Chapters 1 & 2
- Skim "Intermezzo: BSL" from HtDP
  (caveats)