CS 320: Concepts of Programming Languages

Lecture 1: Introduction

Ankush Das

Nathan Mull

Sep 3, 2024

Course Staff

- Instructors: *Nathan Mull* and *Ankush Das*
- Teaching Fellows: *Zachery Casey*, *Qiancheng (Robin) Fu*, and *june wunder*
- No Teaching Assistants
- Course Webpage (Course Outline, Schedule, Important Links, Reading Material): https://nmmull.github.io/CS320/landing/Fall-2024/
- GitHub Repo for Lectures and Assignments: https://github.com/BU-CS320/cs320-fall-2024

Assignment Details

- There will be weekly assignments; both written and programming assignments
- Solutions must be your own! No pair or group submissions permitted.
- Borrowing code snippets from a source like GitHub, StackOverflow, tutorials? You need to add a comment stating the source
- Violations of the above rule would be considered violation of academic integrity and will be reported
- Read more about academic integrity here: https://www.bu.edu/provost/students/undergraduate/academic-integrity/
- No Late Assignment Submissions
- Late Assignment will be given 0 points

Assignment Submissions and Grading

- Submissions via Gradescope (see course webpage)
- Have confusions? Need clarifications? Talk to instructors, TFs, or use *Piazza*
- Re-grade requests must occur within one week of grade release, *always via Gradescope*.
- Consult with TF first. Still have questions? See one of the instructors.
- Monitor Piazza for assignment releases, announcements, etc.
- *HW0*: released today, not graded, just to make sure your system is set up and you can submit on Gradescope

What is this Course About?

- *This is not a course on Programming*! We will not teach "What is programming?"
- This is a course on Programming Languages. We will teach "What is a programming language?"
- More importantly, we will teach you "*Programming Abstractions*". These are high-level concepts (i.e., features) provided by a language that will make it easier to write complex programs.
- Today's Abstraction: *Functions*

What is a Programming Language (PL)?

- You tell me! Have you ever used one before?

What is a Programming Language (PL)?

- You tell me! Have you ever used one before?

- A tool for writing programs and performing computations?

A Programmer's View of PLs

A Programmer's View of PLs

- A tool for programming?
- Stacks / Heaps / Assembly / Bytecode
- Hardware / Registers / Memory

A Programmer's View of PLs

- A tool for programming?
- Stacks / Heaps / Assembly / Bytecode
- Hardware / Registers / Memory

This course is not about any of this!

A PL is so much more!

Programmers are Users of PLs

- An Important Perspective of this Course: programmers are *users* of PLs, not necessarily designers
- e.g., we don't ask people *users* of aeroplanes / trains / cars / road / food to design them, so why PLs
- At the same time, it's important to take their perspective and PL designers should work with programmers to design languages
- So who should design PLs: *mathematicians!!*
- Thankfully, everyone on your course staff is a mathematician!
- By the end of this course, you will also be mathematicians!

A Mathematician's View of PLs

A programming language is a *mathematical object*: like any other mathematical object like numbers, triangles, polynomials, etc.

Alonzo Church

Inventor of today's abstraction: functions

A Mathematician's View of PLs

A programming language is a *mathematical object*: like any other mathematical object like numbers, triangles, polynomials, etc.

Alonzo Church

Inventor of today's abstraction: functions

- Syntax
- Type System
- Semantics

What are these 3 Components?

- **Syntax: How to *write* a program in this PL?**

What are these 3 Components?

- **Syntax: How to *write* a program in this PL?**

```
int x = 2;
```

```
int x = ;2
```

```
x int = 2;
```

What are these 3 Components?

- Syntax: How to *write* a program in this PL?

```
int x = 2;
```

```
int x = ;2
```

```
x int = 2;
```

What are these 3 Components?

- Syntax: How to *write* a program in this PL?

```
int x = 2;
```

```
int x = ;2
```

```
x int = 2;
```

- Type System: What is a *valid program* in this PL?

What are these 3 Components?

- Syntax: How to *write* a program in this PL?

```
int x = 2;
```

```
int x = ;2
```

```
x int = 2;
```

- Type System: What is a *valid program* in this PL?

```
int x = 3 + 5;
```

```
int x = true;
```

```
bool x = 2 + false;
```

What are these 3 Components?

- Syntax: How to *write* a program in this PL?

```
int x = 2;
```

```
int x = ;2
```

```
x int = 2;
```

- Type System: What is a *valid program* in this PL?

```
int x = 3 + 5;
```

```
int x = true;
```

```
bool x = 2 + false;
```

What are these 3 Components?

- Syntax: How to *write* a program in this PL?

```
int x = 2;
```

```
int x = ;2
```

```
x int = 2;
```

- Type System: What is a *valid program* in this PL?

```
int x = 3 + 5;
```

```
int x = true;
```

```
bool x = 2 + false;
```

- Semantics: What should be the *output* of a valid program?

What are these 3 Components?

- Syntax: How to *write* a program in this PL?

```
int x = 2;
```

```
int x = ;2
```

```
x int = 2;
```

- Type System: What is a *valid program* in this PL?

```
int x = 3 + 5;
```

```
int x = true;
```

```
bool x = 2 + false;
```

- Semantics: What should be the *output* of a valid program?

```
if 5 > 0 return 3 else return 4    ------>   3

if 5 > 0 return 3 else return 4    ------>   false
```

What are these 3 Components?

- Syntax: How to *write* a program in this PL?

```
int x = 2;
```

```
int x = ;2
```

```
x int = 2;
```

- Type System: What is a *valid program* in this PL?

```
int x = 3 + 5;
```

```
int x = true;
```

```
bool x = 2 + false;
```

- Semantics: What should be the *output* of a valid program?

```
if 5 > 0 return 3 else return 4    ------>   3
```

```
if 5 > 0 return 3 else return 4    ------>   false
```

Lesson of the Day: What is a PL?

Lesson of the Day: What is a PL?

PL is a mathematical object:
defined using
syntax, type system, and semantics

Today's Programming Abstraction: Functions

$$f(x) = x + x$$

Today's Programming Abstraction: Functions

$$f(x) = x + x$$

$$g(x, y, z) = \text{if } x \text{ then } y \text{ else } z$$

Today's Programming Abstraction: Functions

$$f(x) = x + x$$

$$g(x, y, z) = \text{if } x \text{ then } y \text{ else } z$$

$$h(x, y, z) = \text{if } x \text{ then } f(y) \text{ else } f(z)$$

Today's Programming Abstraction: Functions

$$f(x) = x + x$$

$$g(x, y, z) = \text{if } x \text{ then } y \text{ else } z$$

$$h(x, y, z) = \text{if } x \text{ then } f(y) \text{ else } f(z)$$

$$fib(n) = \text{if } n = 1 \text{ then } 1 \text{ else } fib(n-1) + fib(n-2)$$

Programming Abstraction ≠ PL

$$f(x) = x + x$$

```c
int f(int x) { return x + x; }
```


```
fun f x = x + x
```


```
let f x = x + x;;
```


```
fn f(x : i32) -> i32 { x + x }
```

Functions are in Most PLs

- Important: Functions are a high-level concept, an abstraction that helps us write a piece of code once that can be *reused again and again*
- In fact, the oldest programming language only had one abstraction: *functions*. It was called the λ-calculus!

Functions are in Most PLs

- Important: Functions are a high-level concept, an abstraction that helps us write a piece of code once that can be *reused again and again*
- In fact, the oldest programming language only had one abstraction: *functions*. It was called the λ-calculus!

Alonzo Church

Inventor of λ-calculus

Functions are in Most PLs

- Important: Functions are a high-level concept, an abstraction that helps us write a piece of code once that can be *reused again and again*
- In fact, the oldest programming language only had one abstraction: *functions*. It was called the λ-calculus!

Alonzo Church

Inventor of λ-calculus

λ-calculus

Functions are in Most PLs

- Important: Functions are a high-level concept, an abstraction that helps us write a piece of code once that can be *reused again and again*
- In fact, the oldest programming language only had one abstraction: *functions*. It was called the λ-calculus!

Alonzo Church

Inventor of λ-calculus

λ-calculus

- Function definitions
- Function Applications
- Variables

λ-calculus is Simple Yet Powerful

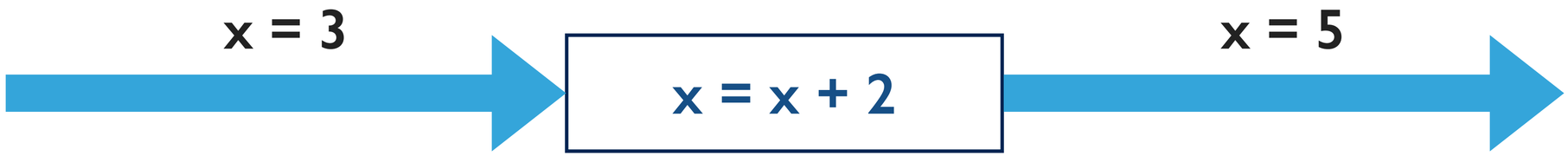
- One of the smallest PLs in the world!
- In contrast, modern PLs have functions, loops, conditionals, arithmetic expressions, booleans, data structures, etc.
- *Every program that can be written in any PL can be written in λ-calculus*
- Alan Turing (Father of Computer Science), who invented the Turing machine, was a PhD student of Alonzo Church
- *Church-Turing Thesis:*
  Every program in Turing machine can be written in λ-calculus and vice-versa

λ-calculus is a Functional PL

- What is a Functional PL?
- Imperative PLs (C, C++, Java, Python) work by updating state by applying operations:
- Functional PLs (OCaml, SML, Haskell) work by calling functions on input to produce output

λ-calculus is a Functional PL

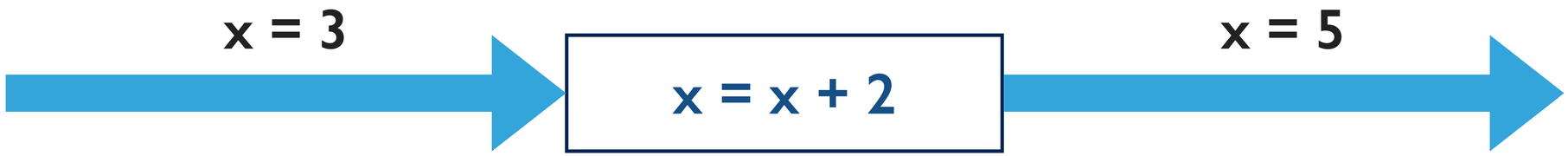
- What is a Functional PL?
- Imperative PLs (C, C++, Java, Python) work by updating state by applying operations:



- Functional PLs (OCaml, SML, Haskell) work by calling functions on input to produce output

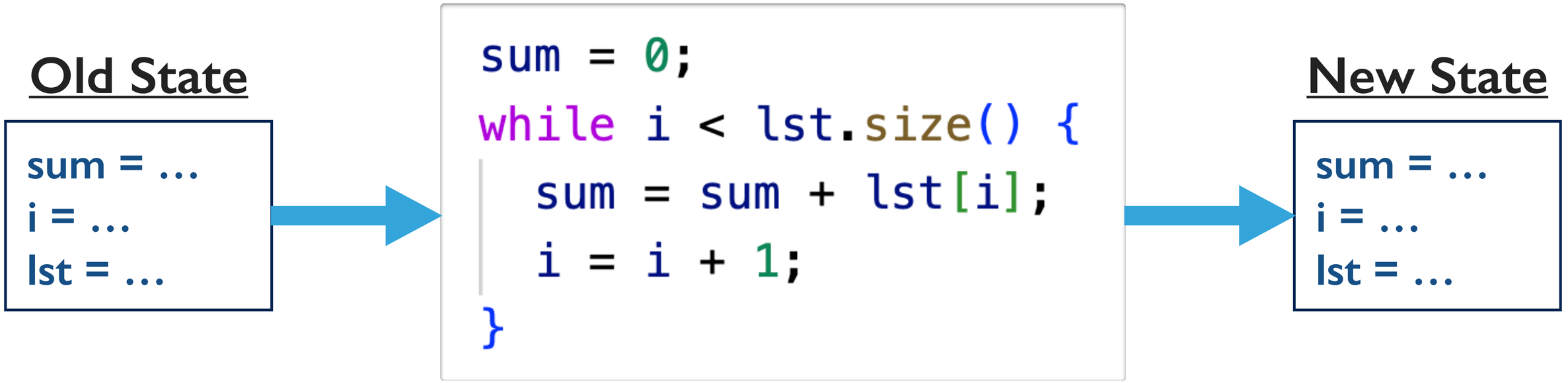
λ-calculus is a Functional PL

- What is a Functional PL?
- Imperative PLs (C, C++, Java, Python) work by updating state by applying operations:



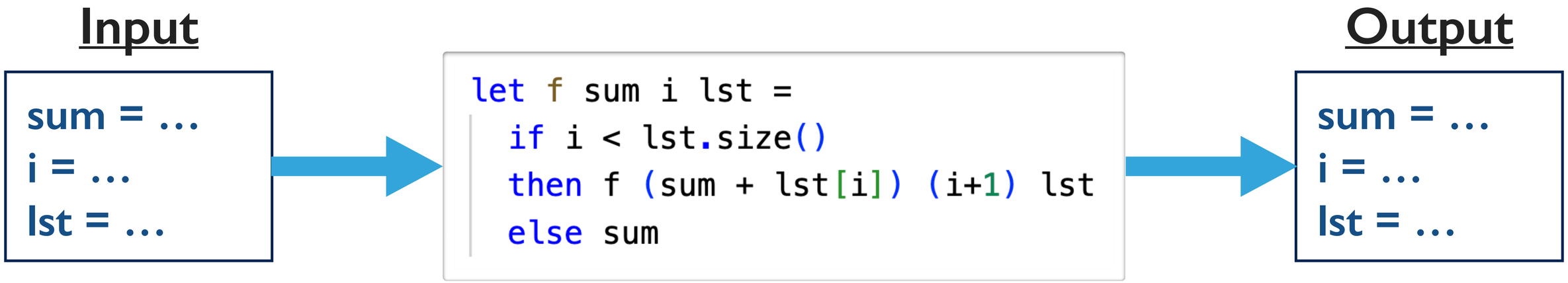
- Functional PLs (OCaml, SML, Haskell) work by calling functions on input to produce output

Sum of a List (Imperative)



- The program defines how to *update old state* at each step
- The program is a set of operations

Sum of a List (Functional)



- The program defines how to *convert input into output*
- The program is just a function

What is a Functional PL?

- A program is just a sequence of functions, each taking some input and producing some output!
- Programmer decides the order in which to apply these functions to get the correct output
- Can every imperative program be written functionally? Yes!
- Can every functional program be written imperatively? Yes!
- Can we convert imperative programs to functional by just passing the entire state as input and getting the entire state as output? Yes!
- But this is very inefficient in practice! The state can be REALLY BIG!

What Functional PL Will We Study?

- Why are we studying functional PLs? They are mathematically well-designed with a syntax, type system, and semantics.
- Which language will we study?

- Why?
  - Easy to use, has a lot of cool mathematical abstractions!
  - Efficient in practice! Has a good community!

What Will We Study in the Course?

- Rest of the Course Will Cover More Fun Programming Abstractions!
  - Recursion
  - Sum and Product Types (Structs and Enums)
  - Higher-Order Programming (Functions with Functions as Inputs)
  - Modules and Monads
- Second Half of the Course: all the above topics will be defined mathematically: using syntax, type system, and semantics

An Important Concluding Point!

- A PL is also a set of programs (functions)
  - Syntax: Implemented by a program called *Parser*
  - Type System: Implemented by a program called *Type Checker*
  - Semantics: Implemented by a program called *Interpreter*
- This course will teach you how to implement all of these!
- *Parser : string ➜ abstract syntax tree*
- *Type Checker : abstract syntax tree ➜ valid / invalid*
- *Interpreter : abstract syntax tree ➜ output*

Next Steps

- Sign up on Piazza and Gradescope! Watch out for announcements on Gradescope.
- Install OCaml on your machines: https://ocaml.org/docs/installing-ocaml
- You should install version 4.13.1, that is the one installed in the Autograder
- Install VS Code: https://code.visualstudio.com/download
- Install the OCaml Platform extension in VS Code, highly recommended!
- Installation issues? Talk to the TFs.

Conclusions: Lessons Learned

- A good PL is a mathematical object: defined using syntax, type system, and semantics.
- Functions are the single most important programming abstraction, present in every major PL, including the oldest one.
- λ-calculus is cool, compact and powerful!
- A PL is a set of functions: parser implements the rules of syntax, type checker implements the rules of type checker, interpreter implements the rules of semantics
- Install OCaml and VS Code! Monitor the course webpage and Piazza.