Comp 311
Functional Programming
Lecture 1

Robert “Corky” Cartwright
Rice University

Robert “Corky” Cartwright

- PhD, semantics and verification of (first–order) functional programs
  - Stanford 1976[1977]
    - Official Advisor: David Luckham
    - Primary Mentor: John McCarthy

- 45+ years of Computer Science research
  - PL theory
    - First-order Programming Logic
    - Semantics of types, sequential functional languages
    - Type systems (Soft typing)

  - PL systems (software engineering)
    - Soft type checker for Scheme
    - Testing concurrent programs
    - DrJava including Functional Java Subset

Course Overview I

- An Introduction to Functional Programming
- Lectures: Tuesdays and Thursdays: 10am – 11:20am
- Office hours: Corky (Online)

  [in normal times Duncan Hall 3104]

  - Tuesdays and Thursdays 1:30pm – 2:30pm
  - By appointment

Course Mechanics

- Course website: Fall 2020 link at https://comp311.rice.edu
  - Syllabus and lectures posted here
  - Lecture topics are subject to change
- Piazza: https://piazza.com/rice/fall2020/comp311
  - Course announcements and Q&A forum
  - Homework assignments and practice exams posted here
- Grading
  - 50% Homework Assignments
  - 25% Mid-term
  - 25% Final
  - Extra credit points on exams, some assignments

Course Overview II

- No required textbook purchase
  - We will draw from a variety of sources including free online textbooks and monographs. Some of them are available for purchase in printed form from online bookstores.

- Coursework consists primarily of weekly homework assignments that are either short programming assignments or written assignments about the underlying theory

- Make sure you do these! They embody the key ideas and principles covered in the course.

Homework Assignments

Think of the programming assignments in this class as very short essays. Focus as much on style as you would for an essay.

50% of a homework grade is based on clarity and style

50% on correctness

Homework Assignments

- Projects are due one week after being assigned.

- Each student has 7 “slip days” to address scheduling conflicts and minor sickness. No more than 3 “slip days” can be used on a given assignment unless you get explicit permission from the instructor.

- Hoard your slip days. The assignments will be progressively more challenging. I predict that some students will not use any slip days.

- Expect to spend about 10 hours outside of class per week.

- Block this time off now in your schedule and respect these commitments.

Homework Assignments

- Assignments are published on Thursdays.
- Start on assignments early so that you have time to ask questions in class, on Piazza, and at office hours.
- A positive attitude and tackling assignments early will help you do your best in the course.

Homework Assignments

- All assignments will be small in scale.
- Most will be given in (the functional subset of) Racket which is a very simple, pure functional language that is easy to simulate in modern type-safe languages like Java, C#, SmallTalk, and Javascript (particularly TypeScript). We will document Racket programs with types. There is an advanced Racket with a "gradual" type system but I disagree with the details if not the spirit.
- We will show how to simulate functional programming in Java, exposing most of the technology used to implement Scala (and perhaps Swift).
- We will also write some programs in Haskell so you are well-equipped to use it as a software engineer.

Homework Assignments

- I strongly recommend that you use the DrRacket programming environment to develop and test Racket programs. The Racket platform runs on Windows, MacOSX, and Linux. If you have a Chromebook, I suggest that you run Linux on it.
- For Java, you have the option of using DrJava or a professional IDE like IntelliJ IDEA or Eclipse but I only use DrJava so I won't be able to answer questions about the professional IDEs.
- I am still researching Haskell platforms but I am leaning toward Visual Studio Code plus a few plugins. IntelliJ is another possibility.
- We will use SVN (turnin on CLEAR) for all assignments.
- Instructions on the course website:
  https://wiki.rice.edu/confluence/display/FPSCALA/Homework+Submission+Guide
  Trick for reaching website if you have forgotten the URL (like I do): enter comp311.rice.edu and follow the Fall 2020 link.

Pause!

What is Functional Programming?

Early Models of Computation

- Turing Machines (Turing)
- Type-0 Grammars (Chomsky)
- The Lambda Calculus (Church)
- Post Machines (Post)

The creators of these models were surprised when they all turned out to be equivalent if computations are confined to functions mapping finite inputs to finite outputs. Now we understand that the notion of computability is an utterly fundamental notion in mathematics.

With exception of Lambda Calculus, all of these models are “bottom-up” frameworks for pushing bits or symbols. But even the Lambda Calculus had a grubby syntactic character because there was no model based on defining and applying functions. It was a vision, an intuition until Scott supplied a truly functional model that could handle self-application and support an isomorphism between D and D -> D.

Early Models of Computation

- Turing Machines (Turing)
- Type-0 Grammars (Chomsky)
- **The Lambda Calculus (Church)**
- *... and many others*
- To the surprise of their inventors, all of these systems turned out to be equivalent in expressive power.
- Suggests there is a deeper structure to the nature of computation.

The Lambda Calculus

- A *calculus* consists of a set of rules for rewriting symbols.
- An attempt to rebuild all of mathematics on the notion of *functions* and *applications.*
- There is no mutation in the lambda calculus.
- Every program consists solely of applications of functions to arguments (which are also functions in the pure lambda calculus, a misleading restriction IMO)
- Applications of functions return values (which are also functions)
- Encoding numbers as functions does not work out well; in the pure lambda calculus, numbers are actually encoded as syntactic descriptions of functions. Equality of functions is undecidable.
- The Pure Lambda Calculus was a critical step in the right direction but it was NOT a true functional programming language. If you add a few constants (nats, suc, if-zero conditional expressions), you get PCF which is a true universal functional programming language. But even PCF is incomplete in fundamental (if practically unimportant) ways.

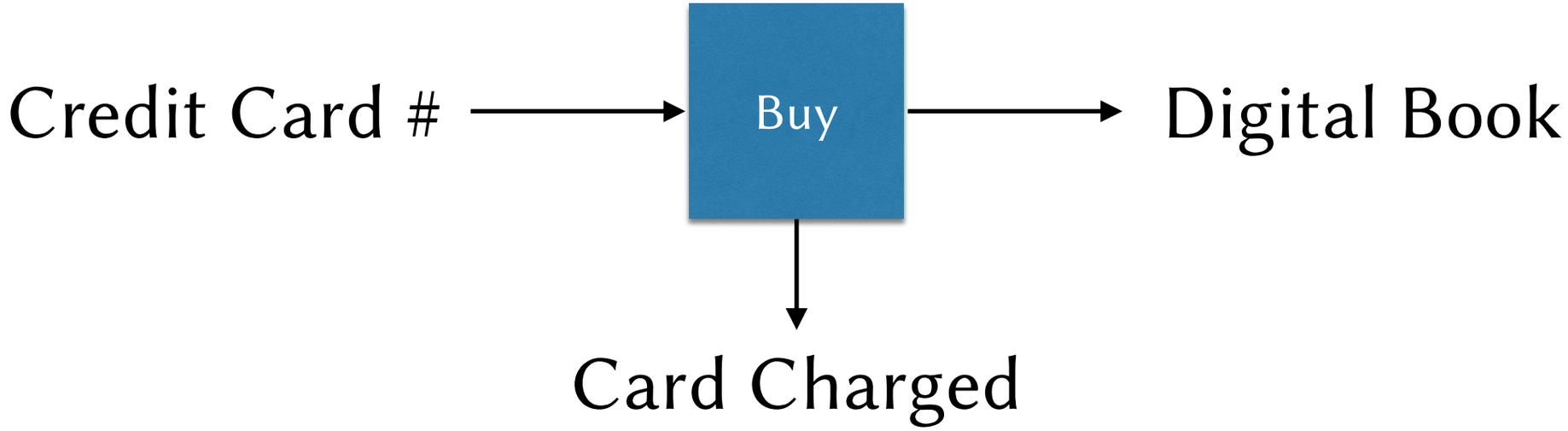
What is Functional Programming?

The Pure Lambda Calculus plus critical constants including the natural numbers.

It is possible (albeit not necessarily desirable) to eliminate variables! Such a combinatory language is unnatural unless

What is Functional Programming?

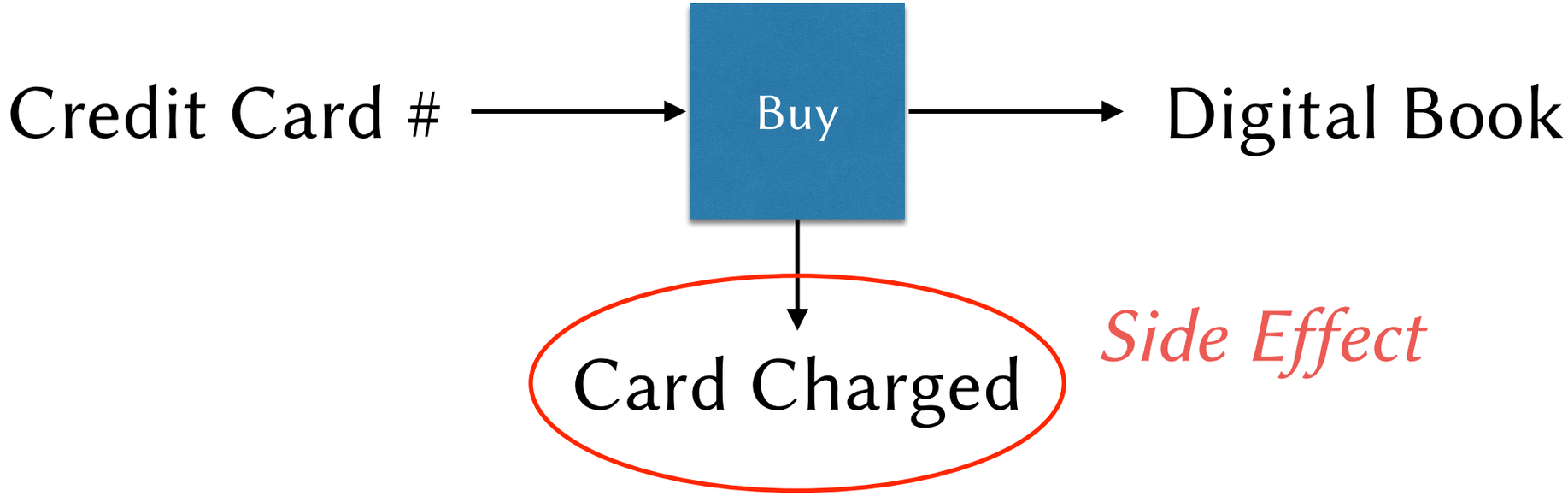
- A style of programming that avoids side effects



- Many applications include operations that are purely functional (no mandated side effects)

What is Functional Programming?

- A style of programming that avoids side effects



What is Functional Programming?

- A style of programming that avoids side effects

b

- All results of a computation are sent as output

Why Avoid Side Effects?

- **Programs are easier to write:** There are fewer interactions between program components, enabling multiple programmers (or a single programmer on multiple days) to work together more easily

- **Programs are easier to read:** Pieces of a program can be read and understood in isolation

- **Programs are easier to test:** Less context needs to be built up before calling a function to test it

- **Programs are easier to debug:** Problems can be isolated more easily, and behavior is inherently deterministic and **local**.

- **Programs are easier to reason about:** The model of computation needed to understand a program without mutation is much simpler; it is ordinary algebra plus induction on the structure of the data.

Why Avoid Side Effects?

- **Programs are easier to execute in parallel:** Because separate pieces of a computation do not interact, it is easy to compute them on separate processors
- This is an increasingly important consideration in the era of multicore chips, big data, and distributing computing
    - *This advantage undermines an often cited argument for mutation (efficiency)*

What is Functional Programming?

- A style of programming that emphasizes functions as the basis of computation
  - Functions are applied to arguments
  - Functions may be passed as arguments to other functions
  - Functions may be returned as values of applications

Pause!

Why Emphasize Functions?

- Functions allow us to factor out common code
  - DRY: Don't Repeat Yourself
  - Why is DRY important?
    - Program understanding
    - Program maintenance
- Passing functions as arguments is often the most straightforward way to abide by DRY
- Returning functions as values is also important for DRY

Why Emphasize Functions?

- Functions allow us to concisely package computations and move them from one control point to another

- Aids us with implementing and reasoning about parallel and distributed programming (yet again)

- Reasoning about sequential programs is easier

  Equational reasoning + induction

A Word on Object-Oriented Programming

- There is no tension between functional and object-oriented programming. In fact, OOP can be cast as an enrichment of FP. See
  https://www.cs.rice.edu/~javaplt/papers/OOPEnrichesFP.pdf

- In many ways, they complement one another.

- Languages like Scala and Swift are designed to integrate both styles of programming

Pause!

Quick Start with Racket

To install Racket on Windows, MacOsX, or Linux,

- Go to https://racket-lang.org/download/ and download the “regular” version of Racket.

- Execute the downloaded installation file.

- Play with Racket arithmetic and simple functions on numbers. Racket performs rational arithmetic until forced to use inexact approximations.