CS 476/MCS 415: Programming Language Design

https://www.cs.uic.edu/~mansky/teaching/cs476/fa23/

Welcome!

- This is CS 476/MCS 415, Programming Language Design
- I'm glad you're here!
- Meets MWF 10:00-10:50 AM in BH 208
- Office hours 2:00 Monday and Friday in SEO 1331 (me), 1:00 Wednesday and 11:00 Thursday in ??? (TA), and by appointment, in-person and on Zoom via Blackboard
  - Office hours are great for homework help!

Course Overview

- Professor: William Mansky (he/him) (mansky1@uic.edu)
- TA: Joseph Wiseman (he/him) (jwisem6@uic.edu)
- Prerequisites: CS 341 (functional programming), CS 151 (logic and proofs)
- Website: https://www.cs.uic.edu/~mansky/teaching/cs476/fa23/
- Anonymous in-class questions: https://pollev.com/wmansky771
- Lectures recorded via Zoom on Blackboard
- Discussion board on Piazza, assignments via Gradescope

Asking questions

- In class: raise your hand anytime
- You can ask questions anonymously with PollEverywhere (https://pollev.com/wmansky771)
- On Piazza
  - Can ask/answer anonymously
  - Can post privately to instructors
  - Can answer other students' questions
- In office hours

- If you have a question, someone else probably has the same question!

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Programming Language Design

- Up till now, you've interacted with PLs as *users*
- We'll look at PLs as *designers* (how should the language work?) and *implementers* (how do we get a computer to run it?)
- We'll look at different kinds of languages and features (imperative, OO, functional, pointers, concurrency, etc.) and figure out how to describe them, and what choices we can make about how they work!

Structure of a language

- Syntax
  - Concrete: what do programs look like?
  - Abstract: what are the pieces of a program?
- Semantics
  - Static: which programs make sense?
  - Dynamic: what do programs do when we run them?
- Pragmatics
  - Implementation: how can we actually make the semantics happen?
  - IDE, tool support, etc.

Metalanguages

- We want to describe how programming languages should work
- We need a *metalanguage*: a language for talking about programming languages

- Metalanguage 1: English (or other natural language)

"The code `x := y + z` sets the value of `x` to the value of `y` plus the value of `z`."

Metalanguages: Natural Language

- Metalanguage 1: English (or other natural language)

"The code `x := y + z` sets the value of `x` to the value of `y` plus the value of `z`."

- Pros: intuitive, familiar, easy to write
- Cons: ambiguous, informal

Metalanguages: Inference Rules

- Metalanguage 2: mathematical logic

$$\frac{(e, \sigma) \Downarrow v}{(x := e, \sigma) \Downarrow \sigma[x \mapsto v]}$$

"If the value of e is v…"

"then $x := e$ sets x to v."

- Pros: precise, formal
- Cons: hard to read and write (we'll work on that!), hard to apply to real programs

Metalanguages: Interpreters

- Metalanguage 3: the OCaml programming language

```
match s with
| Assign x e => update env x (eval env e)
```

- Pros: precise, executable, designed to describe programming languages
- Cons: can be tricky to write (we'll work on that!), has to produce a single answer

Metalanguages

- We want to describe how languages should work, and write code that actually runs those languages
- Natural language: "x := y + z sets x to be y plus z"
- Inference rules: $$\frac{(e, \sigma) \Downarrow v}{(x := e, \sigma) \Downarrow \sigma[x \mapsto v]}$$
- OCaml:
```
match s with
      | Assign x e => update env x (eval env e)
```

- We'll learn to *translate* between these three metalanguages!

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Course outline

- Syntax: grammars, abstract syntax
- Operational semantics and interpreters
- Type systems: checking, inference, safety
- Language types: imperative, functional, OO, logic, …
- Extra features: exceptions, concurrency, …

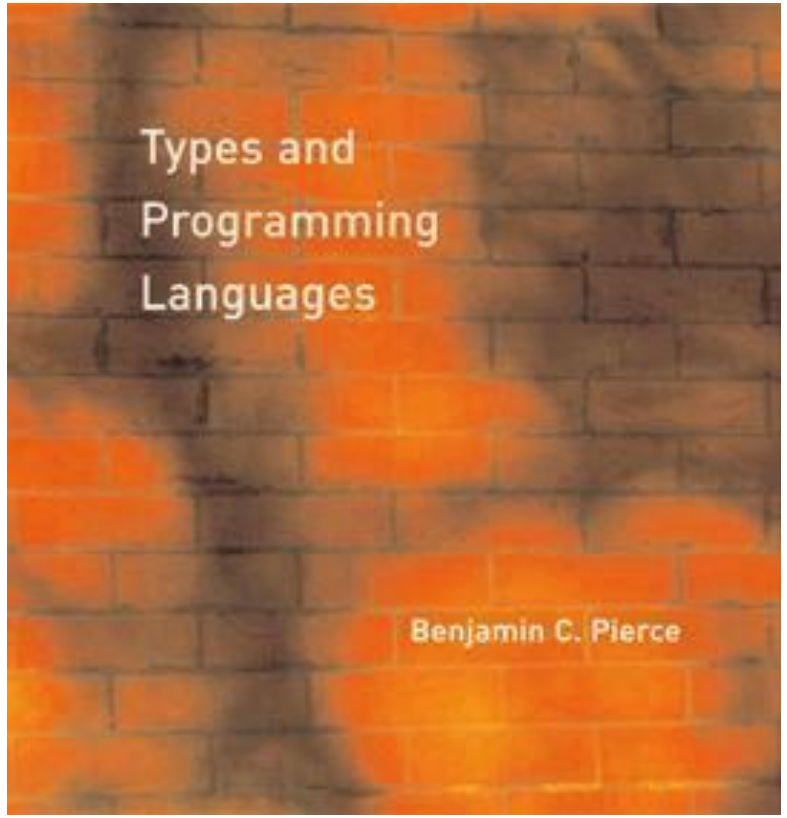
In-Class Exercises

- One question every class, submitted through Gradescope
- Answer them in class if you attend live, or whenever you watch the lecture (within 7 days) if you're watching the recordings
- You don't have to get them right to get credit! Just give your best guess.
- course code Y732V4

- Today's exercise: What's one question about programming languages you'd like to be able to answer by the end of this course?

Textbook

- Types and Programming Languages, Pierce, 2002
- Available online through the library, so you don't need to buy it



Grading

- In-class exercises: 25%
- Assignments: 60%
- Final project: 15%
- Participation: up to 5% extra credit (asking questions in class, posting on Piazza, etc.)

- Final grades will be curved (but only up)

Assignments

- Programming assignments in OCaml: write an interpreter for a language/feature, implement a type checker, etc.
- Written homework: try out logical systems, write proofs about programs
- Each assignment will be submitted twice
  - First submission: write as much as you can; you'll receive full credit as long as you submit anything, and I'll give you feedback
  - Second submission: I'll actually test your code/check your work and grade you on correctness
- Collaboration encouraged, but you must write up your own solution, and cite all sources (websites, collaborators, etc.)
- Submitted and returned via Gradescope (course code Y732V4)

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

The OCaml Programming Language

- OCaml: a functional language in the ML ("metalanguage") family
  - ML family also includes SML, F#, F*, etc.
  - Designed to operate on elements of programming languages

- Strongly-typed functional language with references, based on lambda calculus with pattern-matching

OCaml: The Read-Eval-Print Loop (REPL)

- You can run code without installing at https://try.ocamlpro.com/
- (demo)
- Can also be compiled

HW1 – Getting Started with OCaml

- Posted on the course website
- Set up your OCaml programming environment and write some simple functions in OCaml
- First submission due Thursday 8/24 at 11:59 PM
  - Do as much as you can, get feedback on where you got stuck!
- Submit via Gradescope

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.

Tuples and Functions

```
let p1 = (4, "hi");;
(*  p1 has type “int * string”  *)

let p2 = (3, 5, 2);;
(*  p2 has type “int * int * int”  *)

let incr x y = (x + 1, y + 2);;
(*  incr has type “int -> int -> int * int”  *)

incr 5 6;;
(*  returns (6, 8)  *)
```

Inductive Data Types

- Define a type by giving a list of cases

```
type season = Spring | Summer | Fall | Winter
```

example values: `Summer` `Fall`

```
type value = Intval of int | Stringval of string
           | Floatval of float
```

example values: `Intval 3` `Stringval “hi!”`

```
type intlist = Nil | Cons of int * intlist
```

example values: `Nil` `Cons (1, Nil)`
`Cons (1, Cons (2, Cons (3, Nil)))`

Pattern-Matching and Recursion

```
type season = Spring | Summer | Fall | Winter

let get_temp s =
  match s with
  | Spring -> 70
  | Summer -> 80
  | Fall -> 70
  | Winter -> 30
```

Pattern-Matching and Recursion

```
type value = Intval of int | Stringval of string
           | Floatval of float

let print_val v =
  match v with
  | Intval i ->
  | Stringval s ->
  | Floatval f ->
```

- `i`, `s`, `f` are *new variables* declared in the match cases

Pattern-Matching and Recursion

```
type value = Intval of int | Stringval of string
           | Floatval of float

let print_val v =
  match v with
  | Intval i -> print_int i
  | Stringval s -> print_string s
  | Floatval f -> print_float f
```

- `i`, `s`, `f` are *new variables* declared in the match cases

Pattern-Matching and Recursion

```
type intlist = Nil | Cons of int * intlist

let rec length l =
  match l with
  | Nil -> 0
  | Cons (i, rest) -> length rest + 1
```

Common OCaml Errors

- This expression has type … but is here used with type …

```
let add1 x = x + 1;;
add1 “hi”;;
```

Common OCaml Errors

- This expression has type … but is here used with type …

```
let add1 x = x + 1;;
add1 “hi”;;
```

Error: This expression has type string but an expression was expected of type int

- Think about which of those types is wrong!

Common OCaml Errors

- This expression has type … but is here used with type …

```
let add1 (x : string) = x ^ “1”;;
add1 “hi”;;
(* returns “hi1” *)
```

- Think about which of those types is wrong!

Common OCaml Errors

- This pattern-matching is not exhaustive

```
type value = Intval of int | Stringval of string
           | Floatval of float

let print_val v =
  match v with
  | Intval i -> print_int i
  | Stringval s -> print_string s
```

Warning: this pattern matching is not exhaustive.

Here is an example of a case that is not matched: `Floatval _`

Common OCaml Errors

- This match case is unused

```
type intlist = Nil | Cons of int * intlist

let rec length l =
  match l with
  | Nil -> 0
  | Cons (i, rest) -> length rest + 1
  | Cons (j, rest) -> length rest + 2
```

Common OCaml Errors

- This match case is unused

```
type intlist = Nil | Cons of int * intlist

let rec length l =
  match l with
  | Nil -> 0
  | Cons (i, rest) -> length rest + 1
  | Cons (j, rest) -> length rest + 2
```

Warning: this match case is unused

Common OCaml Errors

- This match case is unused

```
type intlist = Nil | Cons of int * intlist

let rec length l =
  match l with
  | nil -> 0
  | Cons (i, rest) -> length rest + 1
```

Constructors start with capital letters, variables start with lowercase letters!

every argument matches this case

Warning: this match case is unused

Common OCaml Errors

- This expression has type ... but is here used with type ...
- This pattern-matching is not exhaustive
- This match case is unused

- For more, see https://www2.ocaml.org/learn/tutorials/common_error_messages.html

Questions

Nobody has responded yet.

Hang tight! Responses are coming in.