CMSC330: The Expression Problem

Chris Kauffman

Last Updated:
Tue Dec 5 12:26:11 AM EST 2023

Logistics

Reading

The Rust Programming Language

- Official tutorial guide from Rust foundation
- Chapters 1-10 should be sufficient for the course

Goals

- [-] Datatypes and Traits
- [ ] A word on Lifetimes
- [ ] Extras
- [ ] Optional: The Expression Problem

Date	Event
Tue 05-Dec	Rust-wrap
	Expression Problem
Thu 07-Dec	Review Problems
Fri 08-Dec	Dis: Quiz 4
Mon 11-Dec	Project 8 Due
Tue 12-Dec	Reading Day
Wed 13-Dec	Final Exam
	4-6pm
	ESJ 0224

Content Note

- The following topics will NOT be on the exam
- They are very interesting though...

The Expression Problem (Extensibility Problem)

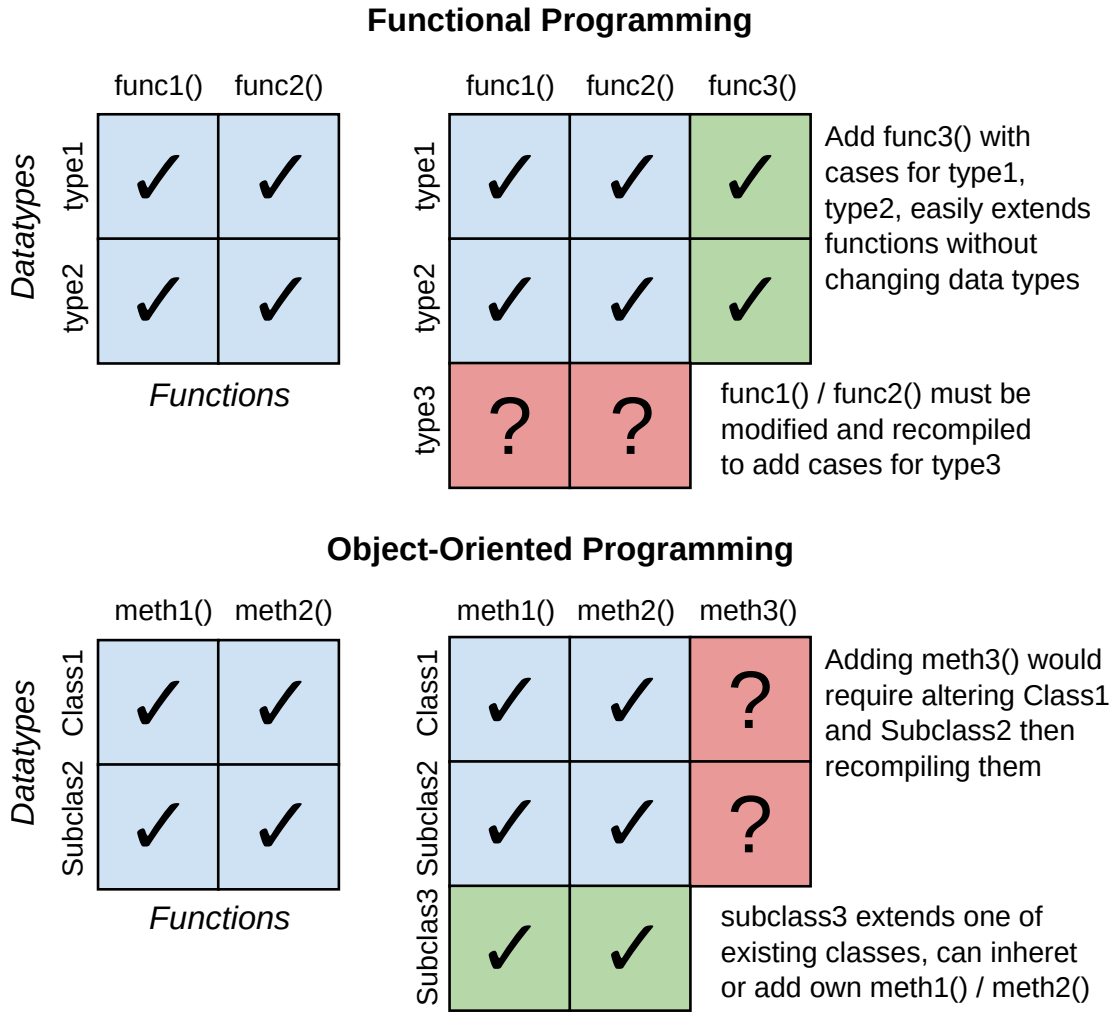
Q: How well can a programming language do these two tasks

(1) Extend Functions

Add a function that works on existing data types without modifying those datatypes

(2) Extend Types

Add a datatype that works with existing functions without changing those functions



A: Traditional Statically Typed Functional and OO Languages favor one or the other task and suffer for the other

Expression Problem in Statically Typed Languages

- Java, OCaml suffer classic symptoms of the Expression Problem: good at either extending functions or datatypes, but not both at once
- Haskell's Type Classes partially solve the Expression Problem[1]
- Rust DOES NOT fully solve the expression problem as it forbids adding `impl` for datatypes outside of the crate in which they are defined (see `extend_string_fail.rs` for an example)
- Likely there are other approaches but the absence of widely known solutions means this may be a limitation of statically typed system

It feels like if Rust lifted the `impl`-within-crate restriction they'd have a full solution but they must have reasons for it...

[1] The inspiration for the grid-based diagram comes from Eli Bendersky's Post about the Expression Problem which provides additional code and detail

Trait Restrictions in Rust

Why does rust restrict implementations of traits to have at least one of "trait in crate" or "type in crate"?

> *This restriction is part of a property called coherence, and more specifically the orphan rule, so named because the parent type is not present. This rule ensures that (A) other peoples code cant break your code and vice versa. Without the rule, (B) two crates could implement the same trait for the same type, and Rust wouldnt know which implementation to use.*
>
> – *The Rust Programming Language, Ch 10.2*

Commentary

- (A) seems false: if Rust's `Iterator` trait were altered to require a `previous()` function as well, all code based on it would break.
- (B) is true, my crate implementing `Iterator` for `i32` could conflict with your crate's implementation of it, so the policy favors preventing potential conflicts over enabling possible convenience

Expression Problem in Dynamically Typed Languages

- Most dynamic languages dodge the Expression Problem as data is open, no compiler to satisfy, allow for dynamic behavior
- Example: Python "Monkey Patching" allows runtime alteration of functions withing classes, addition of new functions, etc.
- **Julia** is Dynamically typed but has many properties similar to Statically Typed languages, features Multiple Dispatch to solve the Expression Problem
- **Clojure** is a dynamically typed language but provides 2 distinct solutions to the Expression Problem: Multimethods and Protocols