Variance in Rust

Covariant, Contravariant, and Invariant

David Wallace Croft, M.Sc.

Presented to the

Indy Rust User Meetup

2024 Jul 03 Wed

Liskov Substitution

- In Java, subclass Car extends superclass Vehicle
- Liskov Substitution Principle
  - Can use a subtype in place of the type

```
final Vehicle vehicle = new Car();
vehicle.setVelocity(0);

static void stop(Vehicle v) { v.setVelocity(0); }
final Boat boat = new Boat();
stop(boat);
```

Inheritance in Rust

- No class inheritance in Rust
  - No subtypes for enums, structs, and unions
  - Not allowed: struct Car extends Vehicle { ... }
- Has something like Java interface inheritance
  - pub trait Car: Vehicle { ... }
- Rust generic types can be "bounded" by traits
  - struct Lease<V: Vehicle>(V);
- Rust lifetimes can have bounds
  - 'b: 'a means 'b lives as least as long as 'a

Trait Bounds

- Car is a subtype of Vehicle

```
pub trait Vehicle {
  fn set_velocity(
    &mut self,
    velocity: f64,
  );
}
```

```
pub trait Car: Vehicle {
  fn get_angle(&self) -> f64;

  fn set_angle(
    &mut self,
    angle: f64,
  );
}
```

Enum Variants

- Enumerated type values are called "variants"
  - Not to be confused with "variance"

```
enum MyEnum {
  MyFirstVariant,
  MySecondVariant,
  MyThirdVariant,
}
```

Unsafe Invariant

- Property that must be upheld for unsafe code
  - A promise that the compiler cannot verify
  - Different from the "invariant" for variance

Non-Comp Sci Definitions

- variance: difference, variation (Latin)
- covariant: two or more things vary together
  - co-: together (Latin)
- contravariant: has an obscure definition in math
  - All of these terms have math definitions
  - contra-: against (Latin)
- invariant: never changing
  - in-: not (Latin)

Computer Science Definitions

- Examples of each of these in following slides

- covariant
  - A subtype can be used in place of a type
- contravariant
  - A supertype can be used in place of a type
- invariant
  - Only the type itself can be used

Covariant

- Can use a subtype in place of the type

```
pub fn stop_vehicle<V: Vehicle>(
  vehicle: &mut V
) {
  vehicle.set_velocity(0.);
}
```

```
pub fn stop_car<C: Car>(
  car: &mut C
) {
  stop_vehicle(car);
}
```

Covariance for Lifetimes

- Can use a subtype in place of the type
- lifetime 'static is a subtype of lifetime 'a
  - 'static can be used wherever 'a is used
- For 'b: 'a, 'b lives at least as long as 'a
  - So 'b is a subtype of 'a

Contravariant

- Can use a supertype in place of the type
  - Only works for fn(T) -> ()

```
pub fn stop<C: Car>(
  c: &mut C,
  f: fn(c: &mut C),
) { f(c); }

pub fn stop_vehicle<V: Vehicle>(
  vehicle: &mut V
) { vehicle.set_velocity(0.); }
```

```
fn test_stop()
{
  let mut car_imp = CarImp {
    velocity: 1.,
  };
  stop(&mut car_imp, stop_vehicle);
  assert_eq!(car_imp.velocity, 0.);
}
```

Contravariance for Lifetimes

- If T is &'static str, &'a str is a supertype

```
pub fn print(
  s: &'static str,
  f: fn(s: &'static str),
) { f(s); }

pub fn print_non_static_str(
  s: &str
) { print!("{s}"); }
```

```
#[test]
fn test_print()
{
  print(
    "static",
    print_non_static_str);
}
```

Invariant

- Cannot use a subtype or supertype for &mut T
  - The code on the right does not compile

```rust
pub fn add_str<'a>(
  v: &mut Vec<&'a str>,
  s: &'a str,
) { v.push(s); }

fn test_add_str_0() {
  let mut v: Vec<&str> = Vec::new();
  let s = String::new();
  add_str(&mut v, &s);
  assert_eq!(v.len(), 1);
}
```

```rust
fn test_add_str_1() {
  let mut v: Vec<&'static str> = Vec::new();

  let s = String::new();

  // Does not compile
  add_str(&mut v, &s);

  assert_eq!(v.len(), 1);
}
```

Links

- "Subtyping and Variance", The Rust Reference, https://doc.rust-lang.org/reference/subtyping.html
- "Subtyping and Variance", The Rustonomicon, https://doc.rust-lang.org/nomicon/subtyping.html
- Jon Gjengset, "Rust for Rustaceans", pp15-16, https://rust-for-rustaceans.com/
- Jon Gjengset, "Crust of Rust: Subtyping and Variance", https://youtu.be/iVYWDIW71jk?si=ty5p8EdKUD0XgWaG

Presenter

- David Wallace Croft, M.Sc.
  - https://www.CroftSoft.com/people/david/
- Organizer of the Dallas Rust User Meetup
  - https://www.DallasRust.org/
- Open source Rust projects
  - Animated interactive games and simulations that run in the browser using WebAssembly (Wasm)
  - Single page applications (SPAs) with static pre-rendering and client-side hydration using Dioxus
  - Serverless functions using Amazon Web Services (AWS) Lambda and Fermyon Spin
  - https://www.CroftSoft.com/people/david/research/rust-wasm/

Licenses

- Slides and code are © 2024 CroftSoft Inc
- This slide presentation is available under the terms of the Creative Commons Attribution 4.0 International License https://creativecommons.org/licenses/by/4.0/
- The code is available under the terms of the open source MIT License https://opensource.org/license/mit/