The state of Rust trying to catch up with Ada

---

whoami

oli-obk

a maintainer of the Rust compiler

---

what is this talk not about

Note: language bashing. on the other hand large corpo bashing is entirely fair game.

---

what it is about

- unsafe/unchecked
- generic packages/modules
- safety certification
- contracts
- subtypes

Note: on everything else I think there's little difference in *capabilites*, even if there are in various important aspects of the usability and error avoidance.

---

unchecked vs unsafe

```
#[no_mangle]
extern "C" fn malloc(_n: usize) -> *const u8 {
    std::ptr::dangling()
}
```

Note: extern/no_mangle not needing unsafe until very recently allowed you to do unsound things without mentioning unsafe

---

generic packages/modules

```
error: expected one of `;` or `{`, found `<`
 --> src/lib.rs:1:8
  |
1 | mod foo<T> {}
  |        ^ expected one of `;` or `{`
```

Note: We support it so little, not even diagnostics know what you meant.

---

safety certification

Ferrocene (https://ferrocene.dev)

> It’s official: Ferrocene is ISO 26262 and IEC 61508 qualified!

Note: After some adacore internal drama where upper leadership is dumb (severing coop with ferrous) that core Ada folk quit (not just over this I think, but timings are curious)

---

contracts

```
use core::contracts::*;

#[requires(x.bar > 50)]
#[ensures(|ret| *ret > 100)]
fn foo(x: Bar) -> i32 {
    x.bar + 50
}
```

https://github.com/rust-lang/rust/pull/128045

Note: very recent impl, not stable

---

pattern types/subtypes

Note: finally we're getting to the real thing of this talk

---

state on stable Rust

```
subtype Non_Zero is Integer range 1..Integer'Last;
subtype Non_Null is not null SomePointer;
```

```
use std::num::NonZeroU32;
use std::ptr::NonNull;
```

Patterns

```
match foo {
    1..100 => {}
    _ => {}
}
```

```
case Foo is
    when 1 .. 100 => null;
    when others => null;
end case;
```

```
match bar {
    Dog | Cat | Bat => {}
    _ => {}
}
```

```
case Bar is
    when Dog | Cat | Bat => null;
    when others => null;
end case;
```

```
subtype Non_Zero is Integer range 1..Integer'Last;
subtype Non_Null is not null SomePointer;
```

```
type NonZeroU32 = u32 is 1..;
// non-null is WIP
type NonNull = *const Thing is !null;
```

So what's the difference?

- pattern types need explicit creation
  - instead of just being part of type conversion
- pattern types do not do strong typing
  - `u32 is 1..` is always the same as any other `u32 is 1..`

- pattern types only coerce, they don't relate

Creation

```
let x: u32 is 1.. = transmute(42);
```

```
subtype Non_Zero is Integer range 1..Integer'Last;
X: Non_Zero := 42;
```

Note: we're discussion allowing directly initializing from literals, but don't hold your breath

---

```
let a: u32 = 42;
let x: u32 is 1.. = transmute(a);
```

```
subtype Non_Zero is Integer range 1..Integer'Last;
A: Integer := 42;
X: Non_Zero := A;
```

Note: we're definitely not allowing this kind of conversion, we don't even allow you to go from `u8` to `u16` silently.

Future creation

Without specifying the pattern again:

```
let a: u32 = 42;
let x: u32 is 1.. = a.try_into().unwrap();
```

Note: via automatically implemented traits

Patterns can be combined

```
type SomePercent = Option<u32> is Some(0..=100);
type Disjunctive = u32 is 0..=100 | 500..=1000;
```

Note: not implemented yet, but "obvious" extension due to patterns allowing this in general

---

Summary

- TODO
  - generic modules
  - initialize pattern types with literals
  - idiomatic conversion from/to pattern types
- WIP
  - contracts
  - pattern types
- caught up
  - safety certification
  - unsafe markings required
- probably not happening
  - strong type aliases