RFMIG: October Session

Specification Languages for Rust

Rust Formal Methods Interest Group — rust-formal-methods.github.io

Etiquette

- Please keep you video switched off. To indicate that you would like to ask a question or speak please switch on your video.
- If you would like to make multiple points or follow-up points please keep your video switched on.
- If your video does not work please use the hand raise reaction in Zoom and keep it activated if you would like to make follow up points.

Schedule

- **November 22** — HACSPEC Case Studies *Bas Spitters*
- **December** — Cancelled for holidays
- **January 24** — MIRAI *Herman Venter*
- **February 28** — Ferrocene *Sabree Blackmon & Florian Gilcher*
- **March 28** — TBA

If you would like to present your work reach out!

Rust Specifications

Subjects to think about

- Manipulating Borrows
- Ownership
- Traits
- Closures
- Interior Mutability
- Unsafe Abstractions

Maximum of Borrows

Let's start small

```
fn max_of<'a>(a: &'a mut u32, b: &'a mut u32) -> &'a mut u32 {
    a.max(b)
}
```

Maximum of Borrows

Prusti

```
#[ensures(*result == old(*a) || *result == old(*b))]
#[ensures(*result >= old(*a) && *result >= old(*b))]
#[ensures(
    if *result == old(*a) {
        after_expiry<'a>(
            *a == before_expiry<'a>(*result)
        )
    } else {
        after_expiry<'a>(
            *b == before_expiry<'a>(*result)
        )
    }
)]
fn max_of<'a>(a: &'a mut u32, b: &'a mut u32) -> &'a mut u32 {
    a.max(b)
}
```

Maximum of Borrows

Creusot

```
#[ensures(result === if *a > *b { a && b.resolve() }
   else { b && a.resolve() })]
fn max_of<'a>(a: &'a mut u32, b: &'a mut u32) -> &'a mut u32 {
    a.max(b)
}
```

Unnesting

This one's trickier

```
fn unnest<'a, 'b : 'a>(x : &'a mut &'b mut u32) -> &'a mut u32 {
  * x
}
```

Unnesting

Prusti

```
#[ensures(
    result == old(*x) &&
    *result == old(**x) &&
    after_expiry<'a>(*old(*x) == before_expiry<'a>(*result))
)]
fn unnest<'a, 'b : 'a>(x : &'a mut &'b mut u32) -> &'a mut u32 {
  * x
}
```

Courtesy of Vytautas Astrauskas (Prusti)

Unnesting

Creusot

```
#[ensures(*result === **x)]
#[ensures(^result === *^x)]
#[ensures(^^x === ^*x)]
fn unnest<'a, 'b : 'a>(x : &'a mut &'b mut u32) -> &'a mut u32 {
  * x
}
```

Courtesy of Xavier Denis (Creusot)

Addition

1 + 1 = 3 ?

```rust
pub trait Add<Rhs = Self> {
    type Output;
    #[must_use]
    fn add(self, rhs: Rhs) -> Self::Output;


}
```

Addition

Prusti

```
#[invariant(forall(|x: Self| x.add(Self::zero()) == x))]
#[invariant(forall(|x: Self, y: Self| x.add(y) == y.add(x)))]
#[invariant(forall(|x: Self, y: Self, z: Self|
   x.add(y.add(z)) == x.add(y).add(z))]
pub trait Add<Rhs = Self> {
    type Output;
    #[must_use]
    fn add(self, rhs: Rhs) -> Self::Output;
    #[ghost]
    fn zero() -> Self;
}
```

```
pub trait Add {
    #[pure]
    #[must_use]
    fn add(self, rhs: Self) -> Self;
    const ZERO : Self::Output;
    #[law]
    #[ensures(x.add(y) === y.add(x))]
    fn commute(x : Self, y: Self);
    #[law]
    #[ensures(x.add(Self::ZERO) === x)]
    fn neutral(x:  Self);
    #[law]
    #[ensures(x.add(y.add(z)) === x.add(y).add(z)
    fn assoc(x: Self, y: Self, z: Self);
}
```

Addition

Creusot

```
pub trait Add {
    #[pure]
    #[must_use]
    fn add(self, rhs: Self) -> Self;

    const ZERO : Self::Output;

    #[law]
    #[ensures(x.add(y) === y.add(x))]
    fn commute(x : Self, y: Self);

    #[law]
    #[ensures(x.add(Self::ZERO) === x)]
    fn neutral(x:  Self);

    #[law]
    #[ensures(x.add(y.add(z)) === x.add(y).add(z))]
    fn assoc(x: Self, y: Self, z: Self);
}
```

Consuming Addition

```
struct Number(u32);
fn consuming_addition(a: Number, b: Number) -> Number {
  Number(a.0 + b.0)
}
```

Consuming Addition

Prusti

```
struct Number(u32);

#[ensures(result.0 == old(a.0) + old(b.0))]
// `old` in this case can also be infered:
#[ensures(result.0 == a.0 + b.0)]
fn consuming_addition(a: Number, b: Number) -> Number {
  Number(a.0 + b.0)
}
```

Consuming Addition

Creusot

```
struct Number(u32);

#[ensures(result.0 === a.0 + b.0)]
fn consuming_addition(a: Number, b: Number) -> Number {
  Number(a.0 + b.0)
}
```