@ Swiss Verification Day, 10 January 2024

Prusti: Deductive Verification for Rust

Vytautas Astrauskas[1], Aurel Bílý[1], Jonáš Fiala[1], Zachary Grannan[2], Christoph Matheja[3], Peter Müller[1], **Federico Poli[1]**, Alexander J. Summers[2]

[1] ETH zürich

[2] THE UNIVERSITY OF BRITISH COLUMBIA

[3] DTU

Reasoning About Imperative Code

```
void client(list *a, list *b)
{
  int old_len = b->len;
  append(a, 100);
  assert(b->len == old_len);
}
```

C

Functional properties

Reasoning About Imperative Code

```
void client(list *a, list *b)
{
  int old_len = b->len;
  append(a, 100);
  assert(b->len == old_len);
}
```

C

Functional properties

Memory Errors

Reasoning About Imperative Code

```c
void client(list *a, list *b)
{
  int old_len = b->len;
  append(a, 100);
  assert(b->len == old_len);
}
```

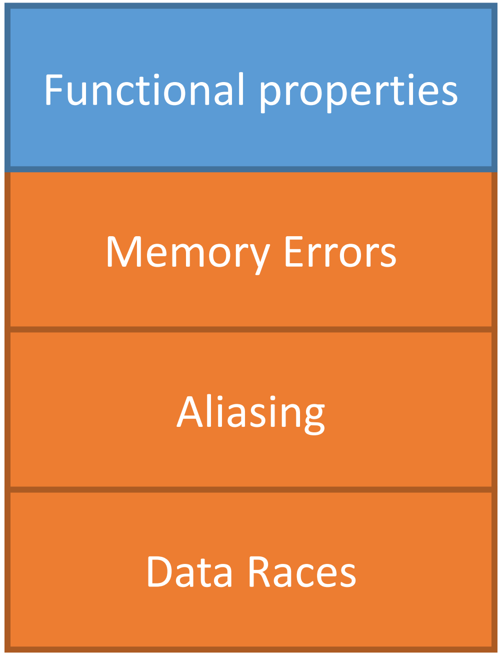
C

- Functional properties
- Memory Errors
- Aliasing

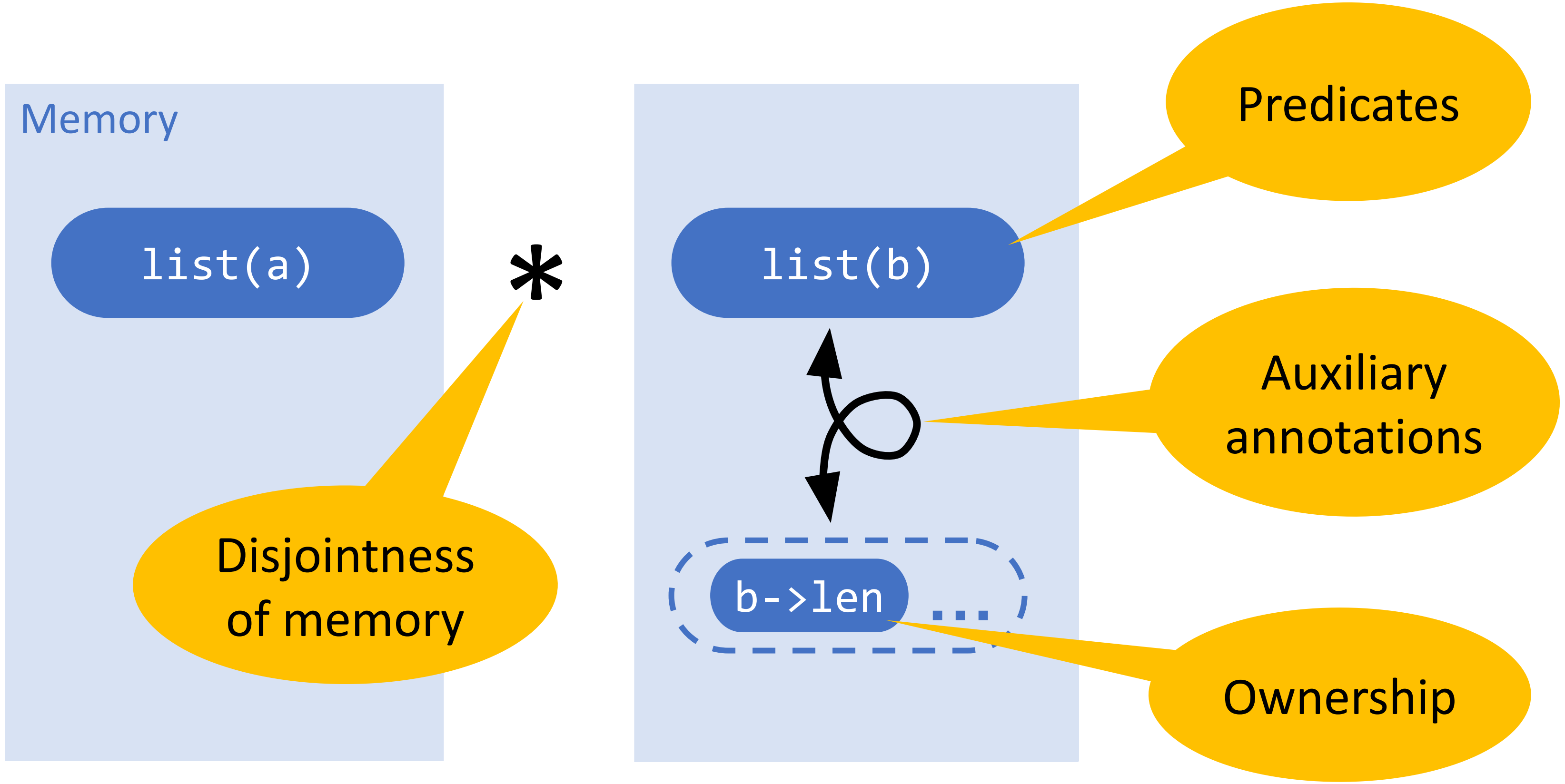
Reasoning About Imperative Code

```c
void client(list *a, list *b)
{
  int old_len = b->len;
  append(a, 100);
  assert(b->len == old_len);
}
```

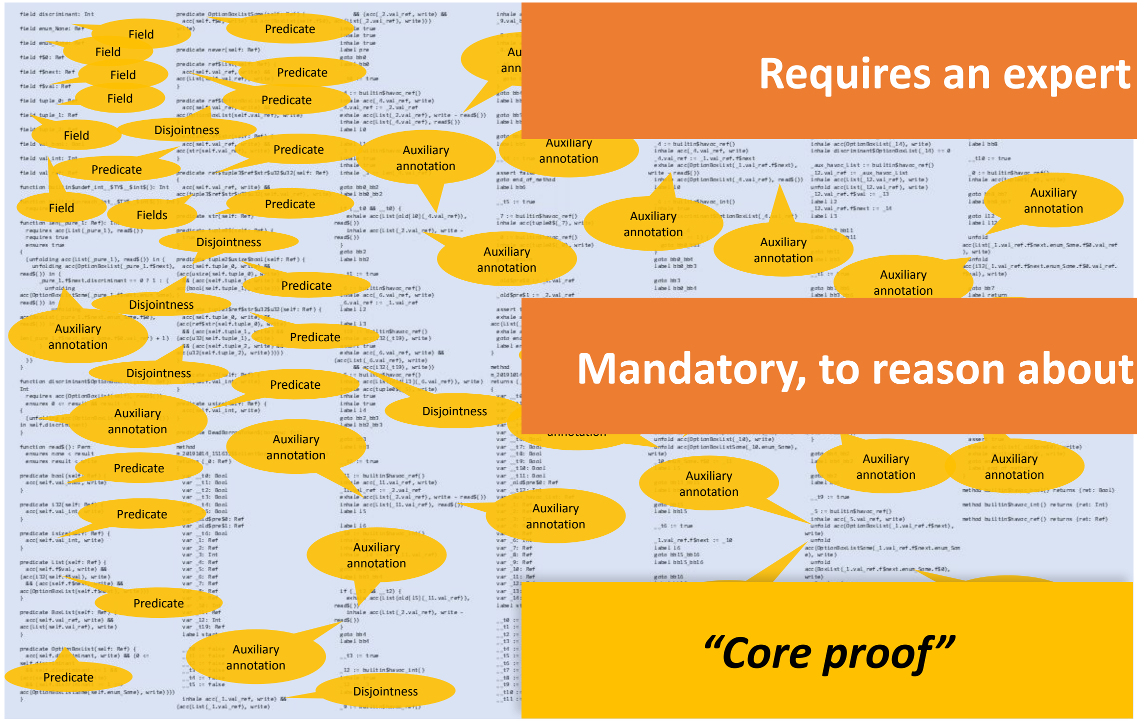
C



Verification Ingredients



Verification Ingredients



Requires an expert

Mandatory, to reason about memory

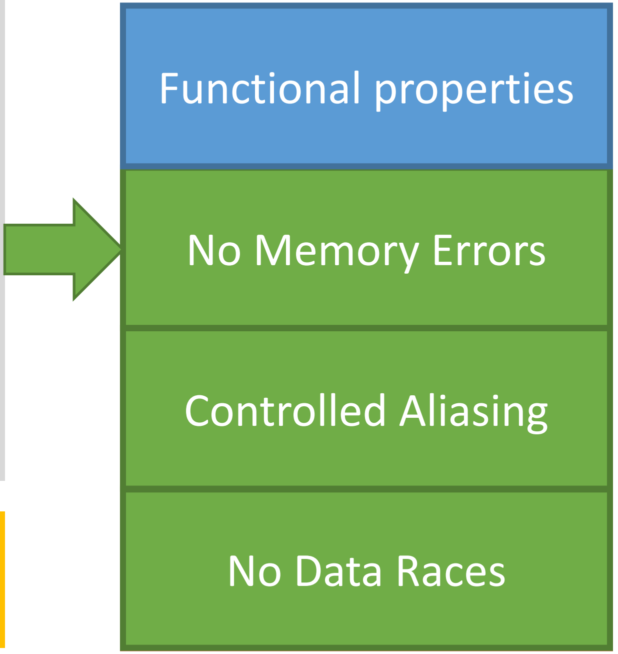
“Core proof”

Rust’s Type System

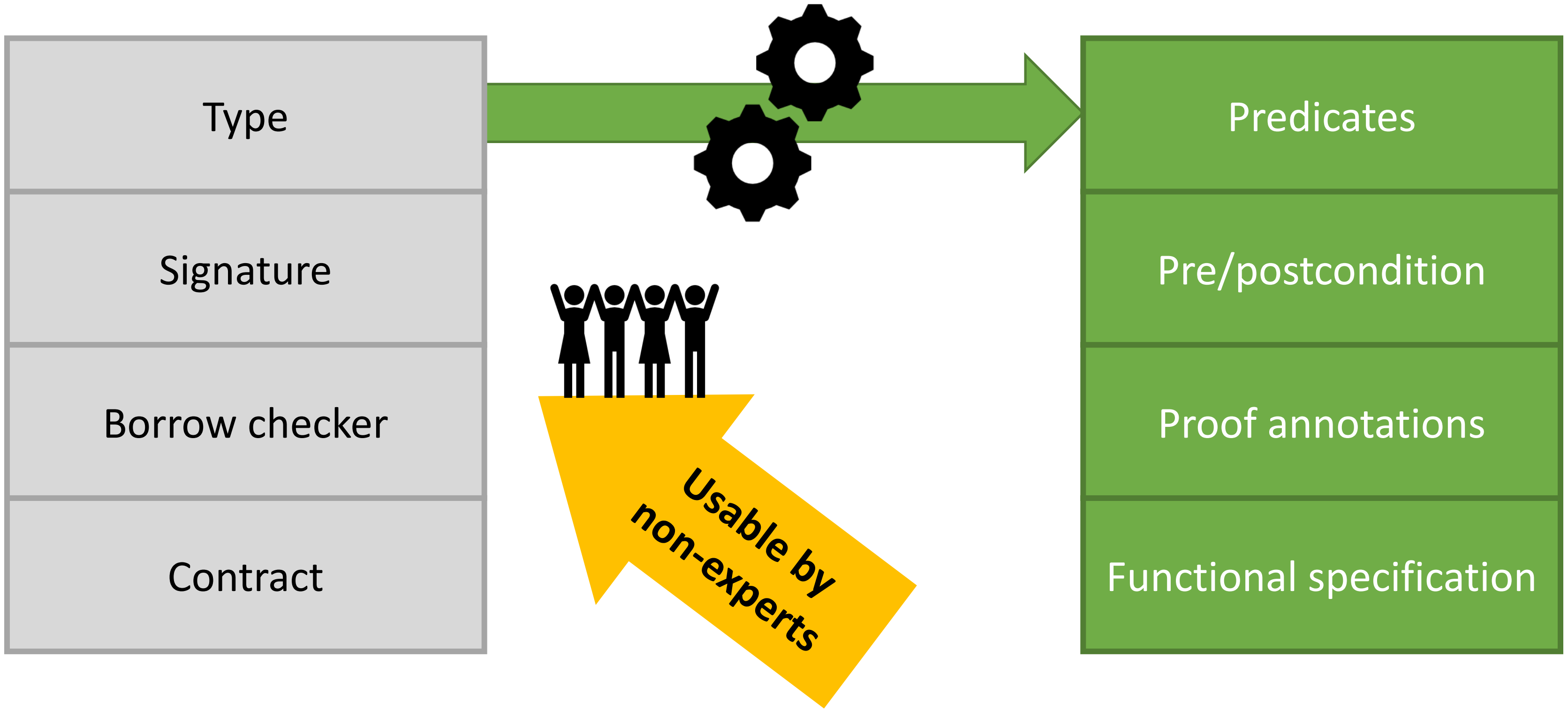
```
fn client(a: &mut List, b: &mut List)
{
  let old_len = b.len();
  append(a, 100);
  assert!(b.len() == old_len);
}
```

Rust

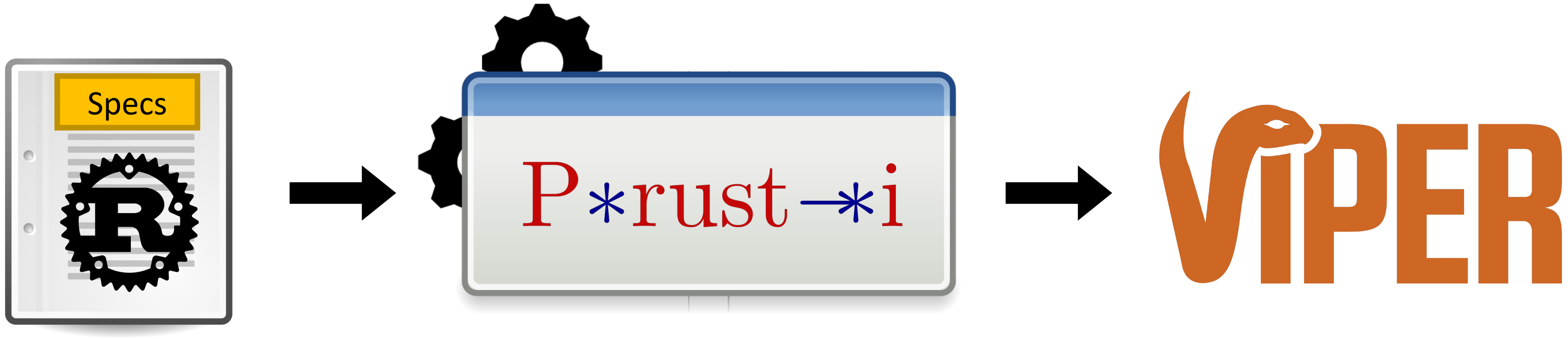
We use the type system to simplify verification



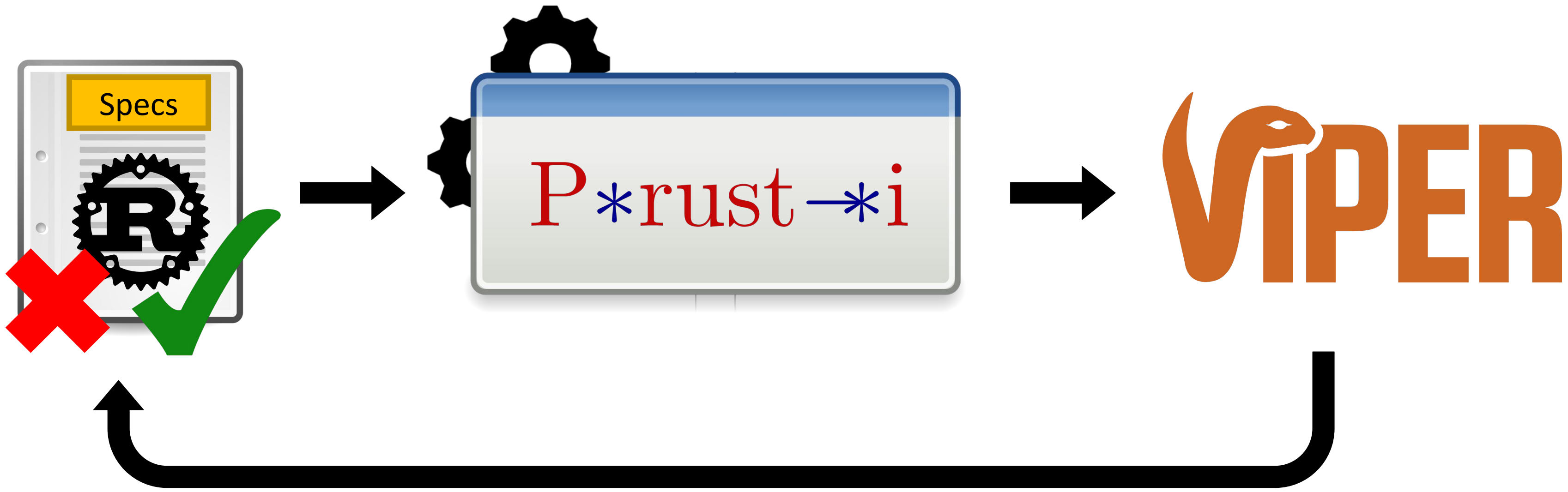
Our Approach



Prusti: An Overview

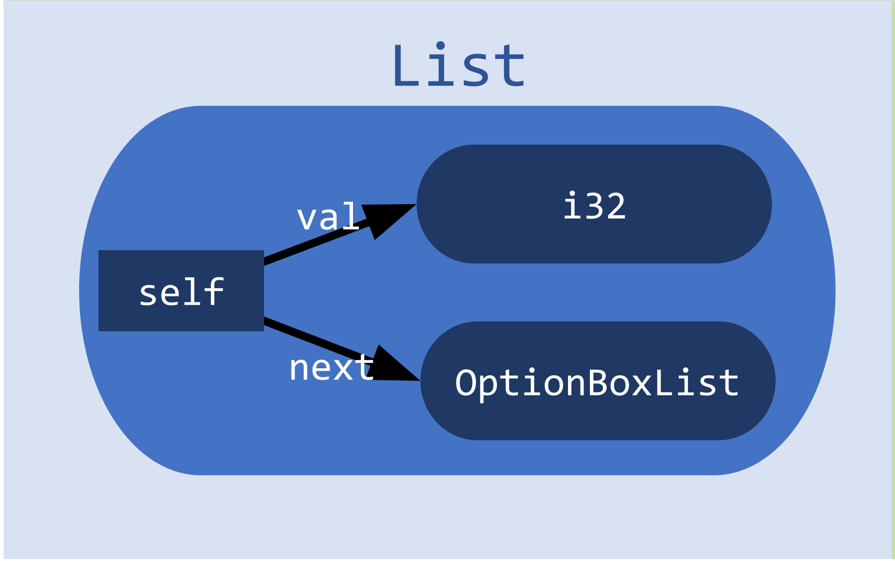


Prusti: An Overview



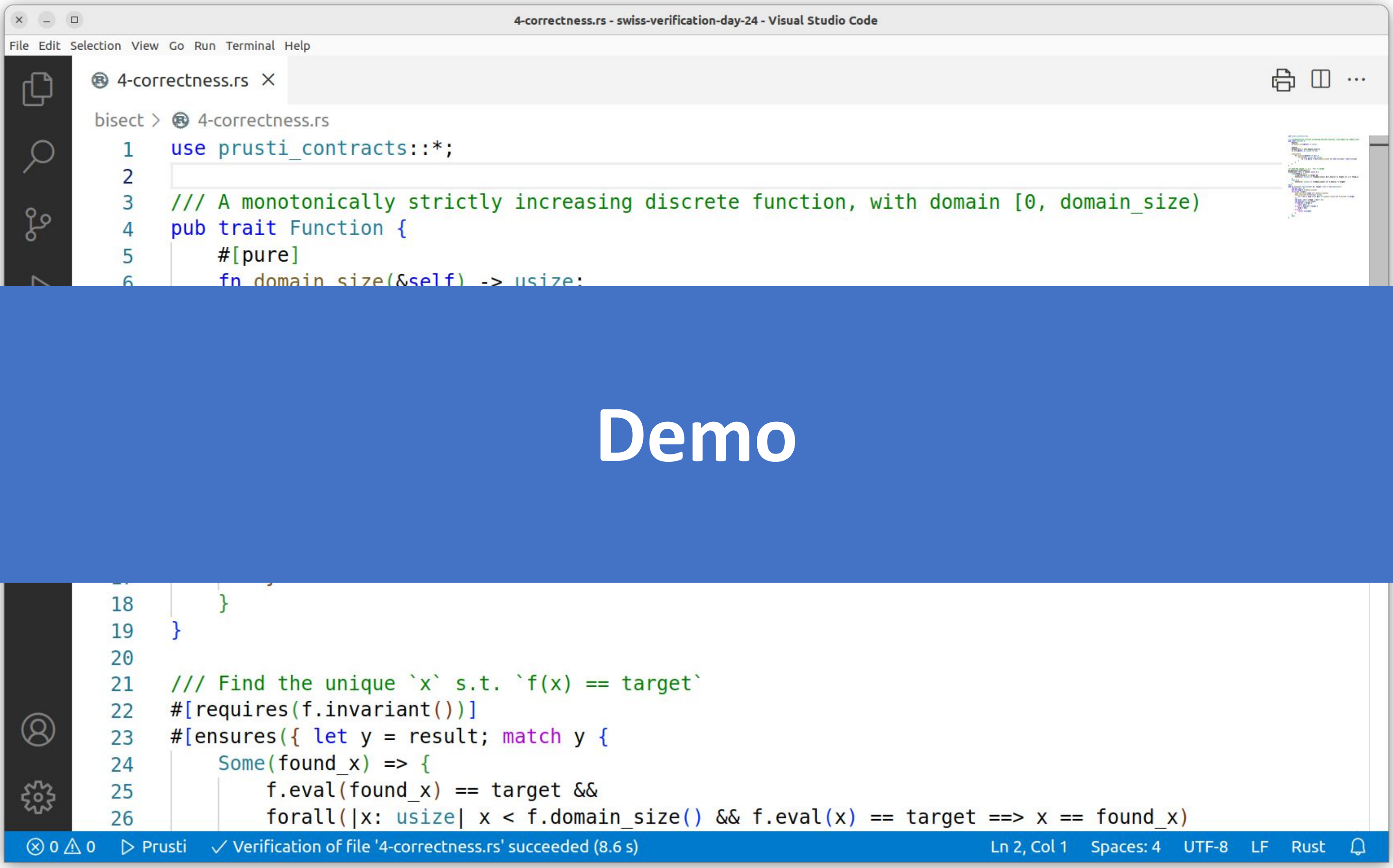
Type Encoding

```
struct List { val: i32, next: Option<Box<List>> }
```



```
predicate List(self: Ref)
{
   acc(self.val) *
   acc(self.next) *
   i32(self.val) *
   OptionBoxList(self.next)
}
```

Viper



4-correctness.rs - swiss-verification-day-24 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
4-correctness.rs
bisect > 4-correctness.rs

```rust
use prusti_contracts::*;

/// A monotonically strictly increasing discrete function, with domain [0, domain_size)
pub trait Function {
    #[pure]
    fn domain_size(&self) -> usize;
```

```rust
    }
}

/// Find the unique `x` s.t. `f(x) == target`
#[requires(f.invariant())]
#[ensures({ let y = result; match y {
    Some(found_x) => {
        f.eval(found_x) == target &&
        forall(|x: usize| x < f.domain_size() && f.eval(x) == target ==> x == found_x)
```

⊗ 0 ⚠ 0 ▷ Prusti ✓ Verification of file '4-correctness.rs' succeeded (8.6 s) Ln 2, Col 1 Spaces: 4 UTF-8 LF Rust

Demo

More Details

VIPER ENCODING

AUTOMATION

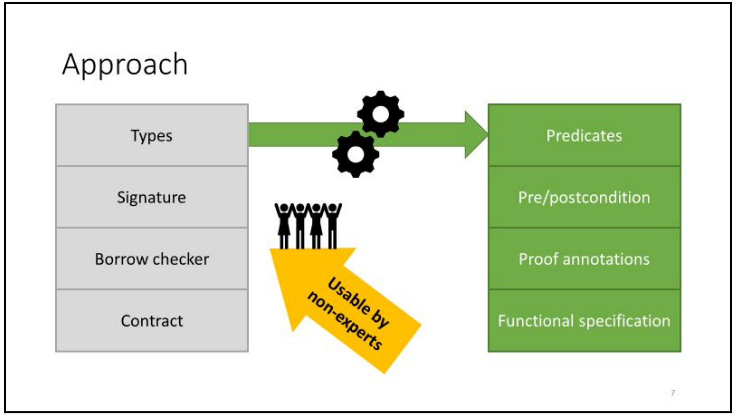
PLEDGES

RUST SUBSET

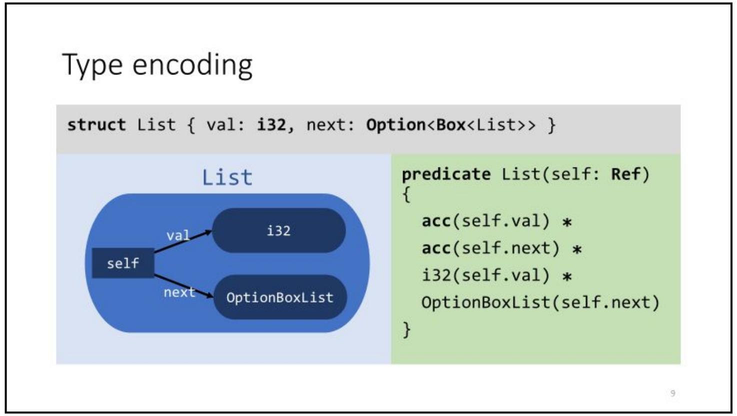
Leveraging Rust Types for Modular Specification and Verification, OOPSLA’19
The Prusti Project: Formal Verification for Rust (invited), NFM’22
Prusti’s user/developer guides

P∗rust–∗i

https://prusti.ethz.ch - https://github.com/viperproject/prusti-dev



Approach



Type encoding

```
struct List { val: i32, next: Option<Box<List>> }
```

```
predicate List(self: Ref)
{
  acc(self.val) *
  acc(self.next) *
  i32(self.val) *
  OptionBoxList(self.next)
}
```

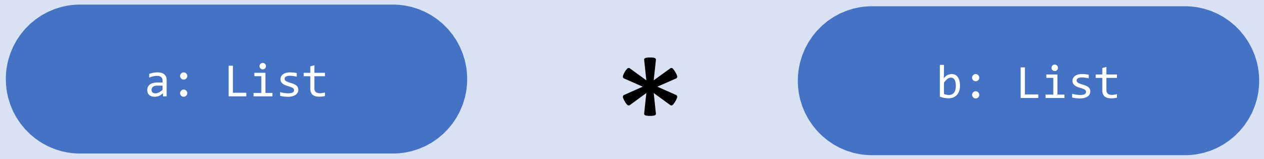
Get in touch with us!

Extra Slides

Signature Encoding

```
fn client(a: &mut List, b: &mut List)
```

Rust



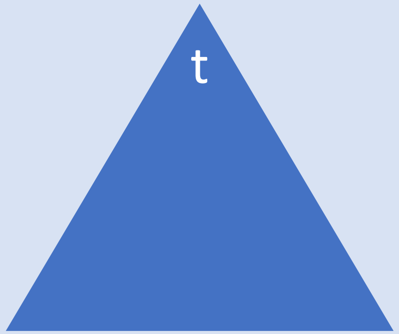
```
method client(a: Ref, b: Ref)
  requires List(a) * List(b)   && a.sorted()
  ensures  List(a) * List(b)   && a.sorted()
```

Viper

Reborrowing Challenges

```rust
fn get(t: &mut BinaryTree) -> &mut BinaryTree {
  t.counter += 1; ... // then return a subtree
}
```

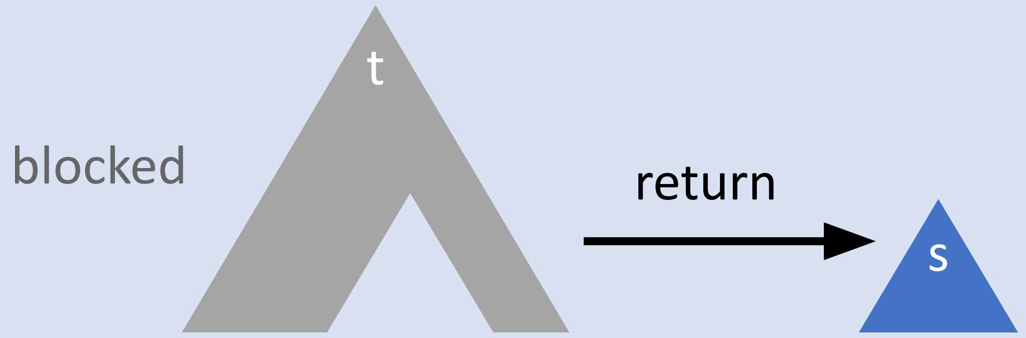
Rust



Reborrowing Challenges

```rust
fn get(t: &mut BinaryTree) -> &mut BinaryTree {
  t.counter += 1; ... // then return a subtree
}
```

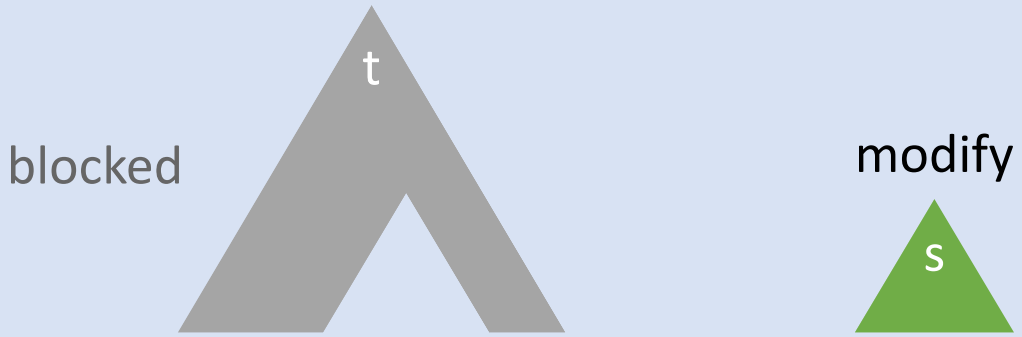
Rust



Reborrowing Challenges

```
fn get(t: &mut BinaryTree) -> &mut BinaryTree {
  t.counter += 1; ... // then return a subtree
}
```

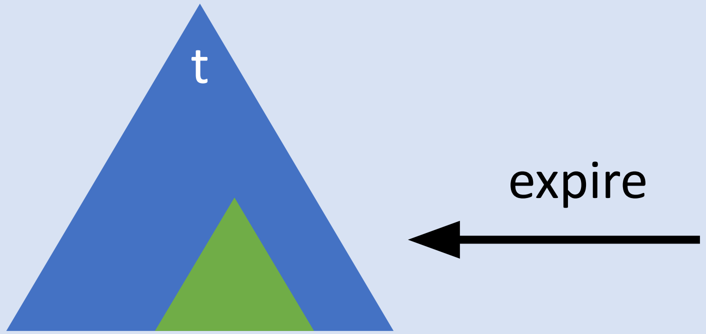
Rust



Reborrowing Challenges

```rust
fn get(t: &mut BinaryTree) -> &mut BinaryTree {
  t.counter += 1; ... // then return a subtree
}
```

Rust



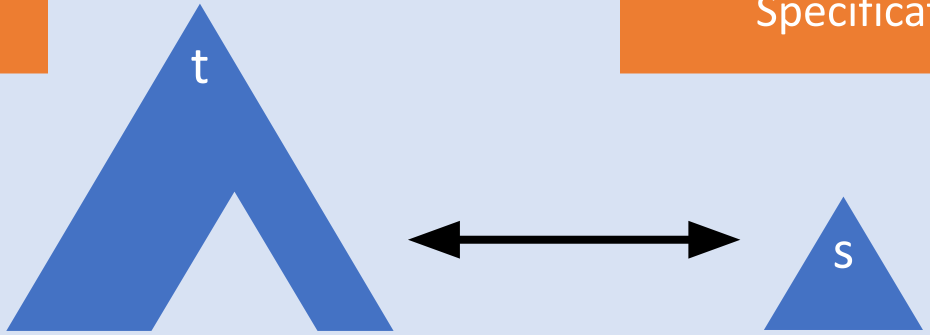
Reborrowing Challenges

```
fn get(t: &mut BinaryTree) -> &mut BinaryTree {
  t.counter += 1; ... // then return a subtree
}
```

Rust

Permissions?

Specification?

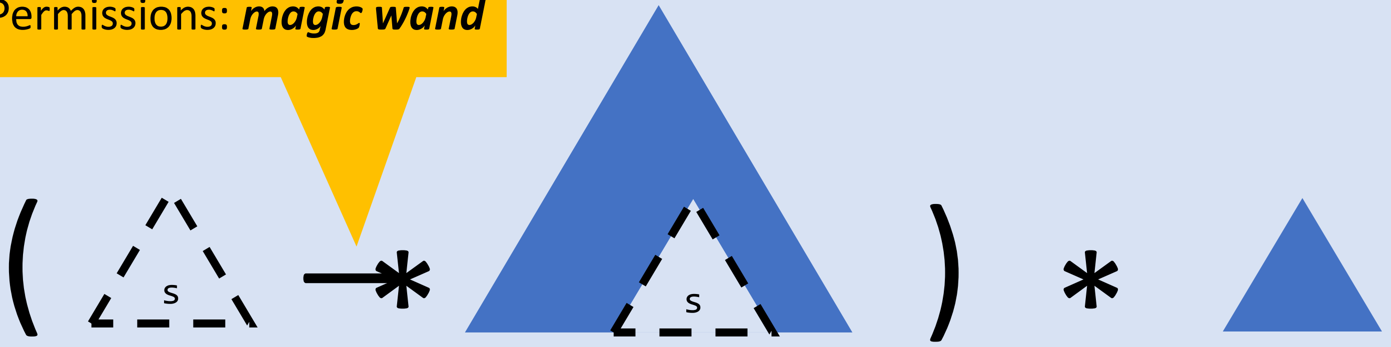


Reborrowing Encoding

```
fn get(t: &mut BinaryTree) -> &mut BinaryTree {
  t.counter += 1; ... // then return a subtree
}
```

Rust

Permissions: ***magic wand***



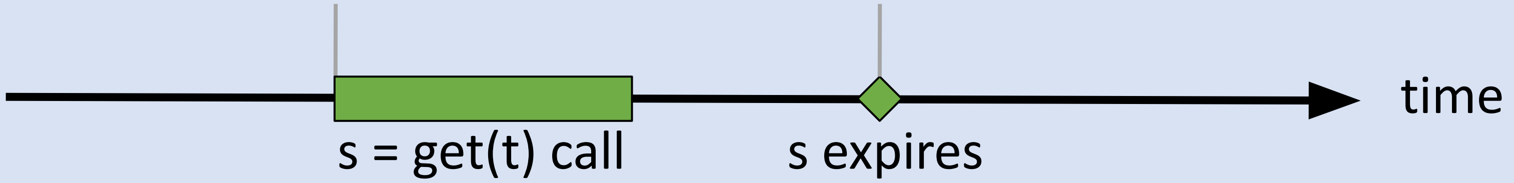
Reborrowing Encoding

```
fn get(t: &mut BinaryTree) -> &mut BinaryTree {
  t.counter += 1; ... // then return a subtree
}
```

Rust

Specification: ***pledge***

ensures: "**old** t.counter" + 1 == "**future** t.counter" …



time
s = get(t) call
s expires