Karlsruher Institut für Technologie

Bitte Logo über `\grouplogo{}` festlegen.

# Corten: Refinement Types for Imperative Languages with Ownership

**Abschlusspräsentation Masterarbeit**

Carsten Csiky | 26th Oktober 2022

# Inhaltsverzeichnis

# Motivation

```
fn max(a: i32, b: i32) {
        if a > b { a } else { b }
}
```

# Motivation

```
fn max(a: i32, b: i32) {
        if a > b { a } else { b }
}
```

Return Value $(v)$ : $v \geq a \wedge v \geq b$

# Motivation

```
fn max(a: i32, b: i32) {
        if a > b { a } else { b }
}
```

Return Value $(v) : v \geq a \wedge v \geq b$

Refinement Types**rondon_liquid_2008** in Functional Programming Languages

# Motivation

```
//@ max(a: i32, b: i32) -> {v:i32 | v >= a && v >= b }
fn  max(a: i32, b: i32) -> i32 {
        if a > b { a } else { b }
}
```

# Motivation

```
//@ max(a: i32, b: i32) -> {v:i32 | v >= a && v >= b }
fn  max(a: i32, b: i32) -> i32 {
        if a > b { a } else { b }
}
```

let $\Gamma = (a : \{v : \texttt{i32} \mid \text{true}\}, b : \{v : \texttt{i32} \mid \text{true}\})$ and $\tau = \{v : \texttt{i32} \mid v \geq a \wedge v \geq b\}$

---

$$\Gamma \vdash \texttt{if}\ a > b\ \{a\}\ \texttt{else}\ \{b\} : \tau$$

# Motivation

```
//@ max(a: i32, b: i32) -> {v:i32 | v >= a && v >= b }
fn  max(a: i32, b: i32) -> i32 {
        if a > b { a } else { b }
}
```

let $\Gamma = (a : \{v : \texttt{i32} \mid \text{true}\}, b : \{v : \texttt{i32} \mid \text{true}\})$ and $\tau = \{v : \texttt{i32} \mid v \geq a \wedge v \geq b\}$

$$\frac{\Gamma, a > b \vdash a : \tau \qquad \Gamma, \neg(a > b) \vdash b : \tau}{\Gamma \vdash \texttt{if } a > b \, \{a\} \texttt{ else } \{b\} : \tau}$$

# Motivation

```
//@ max(a: i32, b: i32) -> {v:i32 | v >= a && v >= b }
fn  max(a: i32, b: i32) -> i32 {
        if a > b { a } else { b }
}
```

let $\Gamma = (a : \{v : \texttt{i32} \mid \text{true}\}, b : \{v : \texttt{i32} \mid \text{true}\})$ and $\tau = \{v : \texttt{i32} \mid v \geq a \wedge v \geq b\}$

$$\dfrac{\dfrac{}{\Gamma, a > b \vdash \{v : \texttt{i32} \mid v \doteq a\} \preceq \tau}}{\Gamma, a > b \vdash a : \tau} \qquad \dfrac{}{\Gamma, \neg(a > b) \vdash b : \tau}$$
$$\Gamma \vdash \texttt{if } a > b \{a\} \texttt{ else } \{b\} : \tau$$

# Motivation

```
//@ max(a: i32, b: i32) -> {v:i32 | v >= a && v >= b }
fn  max(a: i32, b: i32) -> i32 {
        if a > b { a } else { b }
}
```

let $\Gamma = (a : \{v : \texttt{i32} \mid \text{true}\}, b : \{v : \texttt{i32} \mid \text{true}\})$ and $\tau = \{v : \texttt{i32} \mid v \geq a \wedge v \geq b\}$

$$\dfrac{\dfrac{\overset{\star}{\overline{\Gamma, a > b \vdash a : \{v : \texttt{i32} \mid v \doteq a\}}} \qquad \overline{\Gamma, a > b \vdash \{v : \texttt{i32} \mid v \doteq a\} \preceq \tau}}{\Gamma, a > b \vdash a : \tau} \qquad \overline{\Gamma, \neg(a > b) \vdash b : \tau}}{\Gamma \vdash \texttt{if } a > b \, \{a\} \texttt{ else } \{b\} : \tau}$$

# Motivation

```
//@ max(a: i32, b: i32) -> {v:i32 | v >= a && v >= b }
fn  max(a: i32, b: i32) -> i32 {
        if a > b { a } else { b }
}
```

let $\Gamma = (a : \{v : \texttt{i32} \mid \text{true}\}, b : \{v : \texttt{i32} \mid \text{true}\})$ and $\tau = \{v : \texttt{i32} \mid v \geq a \wedge v \geq b\}$

$$\dfrac{\dfrac{\star}{\Gamma, a > b \vdash a : \{v : \texttt{i32} \mid v \doteq a\}} \quad \dfrac{\text{SMT-VALID}\begin{pmatrix}\text{true} \wedge \text{true} \wedge a > b \\ \wedge\, v \doteq a \\ \implies (v \geq a \wedge v \geq b)\end{pmatrix}}{\Gamma, a > b \vdash \{v : \texttt{i32} \mid v \doteq a\} \preceq \tau}}{\Gamma, a > b \vdash a : \tau} \qquad \dfrac{}{\Gamma, \neg(a > b) \vdash b : \tau}$$

$$\Gamma \vdash \texttt{if}\ a > b\ \{a\}\ \texttt{else}\ \{b\} : \tau$$

# Motivation

```
//@ max(a: i32, b: i32) -> {v:i32 | v >= a && v >= b }
fn  max(a: i32, b: i32) -> i32 {
        if a > b { a } else { b }
}
```

let $\Gamma = (a : \{v : \texttt{i32} \mid \text{true}\}, b : \{v : \texttt{i32} \mid \text{true}\})$ and $\tau = \{v : \texttt{i32} \mid v \geq a \wedge v \geq b\}$

$$\dfrac{\dfrac{\star}{\Gamma, a > b \vdash a : \{v : \texttt{i32} \mid v \doteq a\}} \quad \dfrac{\text{SMT-VALID}\begin{pmatrix}\text{true} \wedge \text{true} \wedge a > b \\ \wedge\, v \doteq a \\ \implies (v \geq a \wedge v \geq b)\end{pmatrix}}{\Gamma, a > b \vdash \{v : \texttt{i32} \mid v \doteq a\} \preceq \tau}}{\Gamma, a > b \vdash a : \tau} \quad \dfrac{\vdots}{\Gamma, \neg(a > b) \vdash b : \tau}$$

$$\Gamma \vdash \texttt{if}\ a > b\ \{a\}\ \texttt{else}\ \{b\} : \tau$$

# Motivation

```
fn  clamp(a: &mut i32, b: i32) {
        if *a > b { *a = b }
}
```

# Motivation

```
fn clamp(a: &mut i32, b: i32) {
        if *a > b { *a = b }
}
fn client(...) {
        ...
        clamp(&mut x, 5);
        clamp(&mut y, 6);
        print(x);
        ...
}
```

# Motivation

```
fn clamp(a: &mut i32, b: i32) {
        if *a > b { *a = b }
}
fn client(...) {
        ...
        clamp(&mut x, 5);
        clamp(&mut y, 6);
        print(x);
        ...
}
```

What does this it `print(x)` output?

- In most imperative programming languages:
  - Could be: old `x` or `5`

# Motivation

```
fn clamp(a: &mut i32, b: i32) {
        if *a > b { *a = b }
}
fn client(...) {
        ...
        clamp(&mut x, 5);
        clamp(&mut y, 6);
        print(x);
        ...
}
```

What does this it `print(x)` output?

- In most imperative programming languages:
  - Could be: old `x` or `5`
  - But also `6` (if `x` aliases with `y`)!

# Motivation

```
fn clamp(a: &mut i32, b: i32) {
        if *a > b { *a = b }
}
fn client(...) {
        ...
        clamp(&mut x, 5);
        clamp(&mut y, 6);
        print(x);
        ...
}
```

What does this it `print(x)` output?

- In most imperative programming languages:
  - Could be: old `x` or `5`
  - But also `6` (if `x` aliases with `y`)!
- In Rust:
  - Just old `x` or `5`
  - And nothing else!

# Motivation

```
fn  clamp(a: &mut i32, b: i32) {
        // borrows a
        // owns b
        if *a > b { *a = b }
        // "returns" the borrow of a
}
fn  client(...) { // owns x, y
        ...
        clamp(&mut x, 5); // lend x mutably
        clamp(&mut y, 6); // lend y mutably
        print(x);
        ...
}
```

## Ownership in Rust: Mutability XOR Aliasing

Each lexical scope tracks permissions for visible memory objects. Possible Permission Levels:

- Owner (e.g. `b`)
  - can: read, write
  - transfer ownership (if no outstanding borrows)
- Mutable Reference (e.g. `&mut x`)
  - can: read, write
  - guarantee: no aliasing
- Immutable Reference (e.g. `&v`)
  - can: read, alias
  - guarantee: no mutation

# Motivation

Consequences:

- unique data owner
- no global, mutable state
- no cycles in memory structure

## Ownership in Rust: Mutability XOR Aliasing

Each lexical scope tracks permissions for visible memory objects. Possible Permission Levels:

- Owner (e.g. `b`)
  - can: read, write
  - transfer ownership (if no outstanding borrows)
- Mutable Reference (e.g. `&mut x`)
  - can: read, write
  - guarantee: no aliasing
- Immutable Reference (e.g. `&v`)
  - can: read, alias
  - guarantee: no mutation

# Motivation

Consequences:

- unique data owner
- no global, mutable state
- no cycles in memory structure

Used for:

- safe non-gc memory management
- safe concurrency
- safe low-level hardware access
- ...

## Ownership in Rust: Mutability XOR Aliasing

Each lexical scope tracks permissions for visible memory objects. Possible Permission Levels:

- Owner (e.g. `b`)
  - can: read, write
  - transfer ownership (if no outstanding borrows)
- Mutable Reference (e.g. `&mut x`)
  - can: read, write
  - guarantee: no aliasing
- Immutable Reference (e.g. `&v`)
  - can: read, alias
  - guarantee: no mutation

# Motivation

Consequences:

- unique data owner
- no global, mutable state
- no cycles in memory structure

Used for:

- safe non-gc memory management
- safe concurrency
- safe low-level hardware access
- ...
- $\Rightarrow$ show: program verification as well

## Ownership in Rust: Mutability XOR Aliasing

Each lexical scope tracks permissions for visible memory objects. Possible Permission Levels:

- Owner (e.g. `b`)
  - can: read, write
  - transfer ownership (if no outstanding borrows)
- Mutable Reference (e.g. `&mut x`)
  - can: read, write
  - guarantee: no aliasing
- Immutable Reference (e.g. `&v`)
  - can: read, alias
  - guarantee: no mutation

# Extend Refinement Types to Rust

asd

# Literatur

## Backup-Teil

Folien, die nach `\beginbackup` eingefügt werden, zählen nicht in die Gesamtzahl der Folien.

# Blöcke
## in den KIT-Farben

**Greenblock**
Standard (`block`)

**Blueblock**
= `exampleblock`

**Redblock**
= `alertblock`

**Brownblock**

**Purpleblock**

**Cyanblock**

**Yellowblock**

**Lightgreenblock**

**Orangeblock**

**Grayblock**

**Contentblock**
(farblos)

# Auflistungen

Text

- Auflistung
  Umbruch
- Auflistung
  - Auflistung
  - Auflistung

Bei Frames ohne Titel wird die Kopfzeile nicht angezeigt, und der freie Platz kann für Inhalte genutzt werden.

Bei Frames mit Option `[plain]` werden weder Kopf- noch Fußzeile angezeigt.

# Beispielinhalt

Bei Frames mit Option `[t]` werden die Inhalte nicht vertikal zentriert, sondern an der Oberkante begonnen.

# Beispielinhalt: Literatur

# Farbpalette