Corten: Refinement Types for Imperative Languages with Ownership

Abschlusspräsentation Masterarbeit

Carsten Csiky | 26th Oktober 2022

www.kit.edu

Inhaltsverzeichnis

Motivation
○○○○○○○○

Type System
○○○○○○○○○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○○

2/27   26.10.2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Motivation

```
public IntList square(IntList list) {
  return list.map(x -> x*x);
}
```
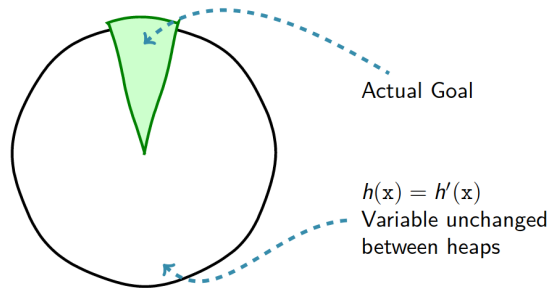

Actual Goal

$h(\mathrm{x}) = h'(\mathrm{x})$
Variable unchanged
between heaps

Motivation
●○○○○○○○

Type System
○○○○○○○○○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○○

3/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Motivation

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

Motivation
○●○○○○○○

Type System
○○○○○○○○○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○○

4/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Motivation

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

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

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work

4/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information Security and Dependability (KASTEL)

Motivation

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

- Return Value $(v) : v \geq a \wedge v \geq b$
- Rondon et al. [RKJ08]: Refinement Types for Functional Programming Languages

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work
○●○○○○○○   ○○○○○○○○○   ○○○   ○○   ○○○

4/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                   Security and Dependability (KASTEL)

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   Type System   Soundness Justification   Related Work   Conclusion / Future Work
○○●○○○○○   ○○○○○○○○○   ○○○   ○○   ○○○

5/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                      Security and Dependability (KASTEL)

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	Type System	Soundness Justification	Related Work	Conclusion / Future Work
○○○●○○○○	○○○○○○○○○	○○○	○○	○○○

6/27 26. 10. 2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

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 $\tau = \{v : \texttt{i32} \mid v \geq a \land v \geq b\}$

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

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 $\tau = \{v : \texttt{i32} \mid v \geq a \wedge v \geq b\}$

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

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work

6/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

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

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

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work

6/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information Security and Dependability (KASTEL)

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

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

Motivation	Type System	Soundness Justification	Related Work	Conclusion / Future Work
○○○●○○○○	○○○○○○○○○	○○○	○○	○○○

6/27   26.10.2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

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

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

$$
\cfrac{}{\Gamma \vdash \text{if } a > b \, \{a\} \text{ else } \{b\} : \tau}
$$

| Motivation | Type System | Soundness Justification | Related Work | Conclusion / Future Work |
| ○○○●○○○○ | ○○○○○○○○○ | ○○○ | ○○ | ○○○ |

6/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information Security and Dependability (KASTEL)

Motivation

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

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work
oooo●ooo   ooooooooo   ooo   oo   ooo

7/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                      Security and Dependability (KASTEL)

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   Type System   Soundness Justification   Related Work   Conclusion / Future Work
○○○○●○○○   ○○○○○○○○○   ○○○   ○○   ○○○

7/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                      Security and Dependability (KASTEL)

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   Type System   Soundness Justification   Related Work   Conclusion / Future Work
○○○○●○○○   ○○○○○○○○○   ○○○   ○○   ○○○

7/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

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   Type System   Soundness Justification   Related Work   Conclusion / Future Work

7/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information Security and Dependability (KASTEL)

Motivation

```rust
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

```rust
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
○○○○○●○○

Type System
○○○○○○○○○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○○

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   Type System   Soundness Justification   Related Work   Conclusion / Future Work

9/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information Security and Dependability (KASTEL)

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   Type System   Soundness Justification   Related Work   Conclusion / Future Work

9/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information Security and Dependability (KASTEL)

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
- . . .
- ⇒ 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

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work

9/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information Security and Dependability (KASTEL)

Contributions

- Empirical Use-Case Analysis
- Refinement Type System
    - Automatic & Decidable Type Checking
    - Path Sensitivity
    - Mutable Data & References
    - Modularity
    - Partial, Mechanized Proof of Soundness
- Implementation
    - Accessible Interface
    - Type-Error Messages with Source Code Locations
    - Counter-Example Generation
- Evaluation
    - Automatic Verification of non-trivial Programs
    - Comparison to other tools

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work
○○○○○○○●   ○○○○○○○○○   ○○○   ○○   ○○○

10/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                                    Security and Dependability (KASTEL)

Contributions

- Empirical Use-Case Analysis
- Refinement Type System
  - Automatic & Decidable Type Checking
  - Path Sensitivity
  - Mutable Data & References
  - Modularity
  - Partial, Mechanized Proof of Soundness
- Implementation
  - Accessible Interface
  - Type-Error Messages with Source Code Locations
  - Counter-Example Generation
- Evaluation
  - Automatic Verification of non-trivial Programs
  - Comparison to other tools

Restrictions

- No Inference System
- Datatypes: Integers, Booleans and References

Type System
Soundness Justification
Related Work
Conclusion / Future Work

Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Overview

- Common Refinement Types
- Mutable Values
- Mutable References
- Function Calls
- Verification of `clamp` Example
- Demonstration

```
fn clamp(
  a: &mut ty!{ a1 : i32 | true => a2 | a2 <= b1 },
  b: ty!{ b1: i32 }
) -> ty!{ v: () } {
  if *a > b {
    *a = b as ty!{ r | (r <= b1) }; ()
  } else {};
  ()
}

fn client() -> ty!{ v: () } {
  ...
  let m = 42;
  clamp(&mut x, m);
  x as ty!{ v : i32 | v < 43 };
}
```

Motivation

Type System

Soundness Justification

Related Work

Conclusion / Future Work

11/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Refinement Types for Rust – Syntax

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

- Embedding using a Macro
- ty!$\{l : b \mid \varphi\}$ in place of a type

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work

12/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Refinement Types for Rust – Syntax

```rust
fn max(
  a: ty!{ av: i32 | true },
  b: ty!{ bv : i32 | true }
) -> ty!{ v : i32 | v >= av && v >= bv } {
  if a > b { a } else { b }
}
```

- Embedding using a Macro
- ty!$\{l : b \mid \varphi\}$ in place of a type

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work

12/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                          Security and Dependability (KASTEL)

Refinement Types for Rust – Syntax

```
fn max(
  a: ty!{ av: i32 },
  b: ty!{ bv : i32 }
) -> ty!{ v : i32 | v >= av && v >= bv } {
  if a > b { a } else { b }
}
```

- Embedding using a Macro
- ty!$\{l : b \mid \varphi\}$ in place of a type

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work

12/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information Security and Dependability (KASTEL)

Type Updates

```rust
fn incr() -> ty!{ w : i32 | w > 0 } {
 let mut i = ... as ty!{ v: i32 | v >= 0 };
 i = i + 1;
 i
}
```

- Types need to change through execution
  - $\Rightarrow$ Type Updates
  - $\Gamma \vdash s \Rightarrow \Gamma'$ (Statement Type Checking)
  - $\Gamma \vdash e : \tau$ (Expression Typing)

Motivation
○○○○○○○○

Type System
○○●○○○○○○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○○

13/27   26.10.2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Type Updates

```
fn incr() -> ty!{ w : i32 | w > 0 } {
 let mut i = ... as ty!{ v: i32 | v >= 0 };
 i = i + 1;
 i
}
```

- Type of `i` after decrementing?
  - Naïve: `ty!{ v : i32 | v = v + 1 }`
- How to keep type context consistent?
  - separation of program-variables and logic-variables
  - Γ: association of program- to logic-variables and predicate
  - on assignment: *replace* association, *append* predicate
  - observation: assignments can not invalidate existing predicates

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work

13/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                           Security and Dependability (KASTEL)

Type Updates

```
fn incr() -> ty!{ w : i32 | w > 0 } {
  // Γ₁ = ({} , true)
  let mut i = ... as ty!{ v: i32 | v >= 0};
  // Γ₂ = ({i ↦ v} , v > 0)
  i = i + 1;
  // Γ₃ = ({i ↦ v₂} , v > 0 ∧ v₂ ≐ v + 1)
  i }
```

- Type of `i` after decrementing?
    - Naïve: `ty!{ v : i32 | v = v + 1 }`
- How to keep type context consistent?
    - separation of program-variables and logic-variables
    - Γ: association of program- to logic-variables and predicate
    - on assignment: *replace* association, *append* predicate
    - observation: assignments can not invalidate existing predicates

Type Updates

```
fn incr() -> ty!{ w : i32 | w > 0 } {
  // Γ₁ = ({}, true)
  let mut i = ... as ty!{ v: i32 | v >= 0};
  // Γ₂ = ({i ↦ v}, v > 0)
  i = i + 1;
  // Γ₃ = ({i ↦ v₂}, v > 0 ∧ v₂ ≐ v + 1)
  i }
```

$$\text{INTRO-SUB} \frac{\Gamma \vdash e : \tau \qquad \Gamma \vdash \tau \preceq \tau'}{\Gamma \vdash e \text{ as } \tau' : \tau'}$$

$$\text{DECL} \frac{\Gamma \vdash e : \{\beta : b \mid \varphi\}}{\Gamma \vdash \text{let } x = e \Rightarrow \Gamma[x \mapsto \beta], \varphi}$$

Motivation
Type System
Soundness Justification
Related Work
Conclusion / Future Work

13/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information Security and Dependability (KASTEL)

Type Updates

```
fn incr() -> ty!{ w : i32 | w > 0 } {
  // Γ₁ = ({}, true)
  let mut i = ... as ty!{ v: i32 | v >= 0};
  // Γ₂ = ({i ↦ v}, v > 0)
  i = i + 1;
  // Γ₃ = ({i ↦ v₂}, v > 0 ∧ v₂ ≐ v + 1)
  i }
```

$$\text{BINOP} \quad \frac{\Gamma \vdash \alpha \text{ fresh}}{\Gamma \vdash x_1 \odot x_2 : \{\alpha : b \mid \alpha \simeq [\![x_1 \odot x_2]\!]\Gamma\}}$$

$$\text{ASSIGN} \quad \frac{\Gamma \vdash e : \{\beta : b \mid \varphi\}}{\Gamma \vdash x = e \Rightarrow \Gamma[x \mapsto \beta], \varphi}$$

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work

13/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information Security and Dependability (KASTEL)

Type Updates

```
fn incr() -> ty!{ w : i32 | w > 0 } {
  // Γ₁ = ({}, true)
  let mut i = ... as ty!{ v: i32 | v >= 0};
  // Γ₂ = ({i ↦ v}, v > 0)
  i = i + 1;
  // Γ₃ = ({i ↦ v₂}, v > 0 ∧ v₂ ≐ v + 1)
  i }
```

$$\text{SEQ } \frac{\Gamma \vdash s_1 \Rightarrow \Gamma' \qquad \Gamma' \vdash s_2 \Rightarrow \Gamma''}{\Gamma \vdash s_1; s_2 \Rightarrow \Gamma''}$$

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work
○○○○○○○○   ○○●○○○○○○   ○○○   ○○   ○○○

13/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                                        Security and Dependability (KASTEL)

References – Strong Updates

```
fn client() -> ty!{ v: i32 | v == 4 } {
  let mut a = 2;        // a : {v₁ : i32 | v₁ == 2}
  let mut q = 3;        // q : {v₂ : i32 | v₂ == 3}
  let mut b = &mut a;   // b : {v₃ : &i32 | v₃ == &a}
  *b = 0;               // changes a's value and type
  b = &mut q;           // b : {v₄ : &i32 | v₄ == &q}
  *b = 4;               // changes q's value and type
  a
}
```

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work
○○○○○○○○   ○○○●○○○○○   ○○○   ○○   ○○○

14/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                               Security and Dependability (KASTEL)

References – Strong Updates

```
fn client() -> ty!{ v: i32 | v == 4 } {
  let mut a = 2;          // a : {v₁ : i32 | v₁ == 2}
  let mut q = 3;          // q : {v₂ : i32 | v₂ == 3}
  let mut b = &mut a;     // b : {v₃ : &i32 | v₃ == &a}
  *b = 0;                 // changes a's value and type
  b = &mut q;             // b : {v₄ : &i32 | v₄ == &q}
  *b = 4;                 // changes q's value and type
  a
}
```

$$\text{L\tiny IT} \; \frac{\Gamma \vdash \alpha \text{ fresh}}{\Gamma \vdash v : \{\alpha : b \mid \alpha \simeq [\![v]\!]\Gamma\}}$$

Motivation
○○○○○○○○

Type System
○○○●○○○○○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○○

14/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                          Security and Dependability (KASTEL)

```
fn client() -> ty!{ v: i32 | v == 4 } {
  let mut a = 2;        // a : {v₁ : i32 | v₁ == 2}
  let mut q = 3;        // q : {v₂ : i32 | v₂ == 3}
  let mut b = &mut a;   // b : {v₃ : &i32 | v₃ == &a}
  *b = 0;               // changes a's value and type
  b = &mut q;           // b : {v₄ : &i32 | v₄ == &q}
  *b = 4;               // changes q's value and type
  a
}
```

$$\text{REF} \frac{\Gamma \vdash \alpha \text{ fresh}}{\Gamma \vdash \&x : \{\alpha : \&b \mid \alpha \simeq [\![\&x]\!]\Gamma\}}$$

Motivation
○○○○○○○○

Type System
○○○●○○○○○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○○

14/27   26.10.2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

References – Strong Updates

```
fn client() -> ty!{ v: i32 | v == 4 } {
  let mut a = 2;          // a : {v₁ : i32 | v₁ == 2}
  let mut q = 3;          // q : {v₂ : i32 | v₂ == 3}
  let mut b = &mut a;     // b : {v₃ : &i32 | v₃ == &a}
  *b = 0;                 // changes a's value and type
  b = &mut q;             // b : {v₄ : &i32 | v₄ == &q}
  *b = 4;                 // changes q's value and type
  a
}
```

$$\text{ASSIGN-STRONG} \quad \frac{\Gamma(z) = \beta \qquad \Gamma \vdash x \in \{\&y\} \qquad \Gamma \vdash \gamma \text{ fresh}}{\Gamma \vdash *x = z \Rightarrow \Gamma[y \mapsto \gamma], \gamma \doteq \beta}$$

(Also ASSIGN-WEAK)

Motivation
○○○○○○○○

Type System
○○○●○○○○○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○○

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

References – Strong Updates

```
fn client() -> ty!{ v: i32 | v == 4 } {
  let mut a = 2;        // a : {v₁ : i32 | v₁ == 2}
  let mut q = 3;        // q : {v₂ : i32 | v₂ == 3}
  let mut b = &mut a;   // b : {v₃ : &i32 | v₃ == &a}
  *b = 0;               // changes a's value and type
  b = &mut q;           // b : {v₄ : &i32 | v₄ == &q}
  *b = 4;               // changes q's value and type
  a
}
```

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work
○○○○○○○○   ○○○●○○○○○   ○○○   ○○   ○○○

14/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                                  Security and Dependability (KASTEL)

References – Strong Updates

```
fn client() -> ty!{ v: i32 | v == 4 } {
  let mut a = 2;          // a : {v₁ : i32 | v₁ == 2}
  let mut q = 3;          // q : {v₂ : i32 | v₂ == 3}
  let mut b = &mut a;     // b : {v₃ : &i32 | v₃ == &a}
  *b = 0;                 // changes a's value and type
  b = &mut q;             // b : {v₄ : &i32 | v₄ == &q}
  *b = 4;                 // changes q's value and type
  a
}
```

$$\text{Var} \frac{\Gamma \vdash \alpha \text{ fresh}}{\Gamma \vdash x : \{\alpha : b \mid \alpha \simeq [\![x]\!]\Gamma\}}$$

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work

14/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information Security and Dependability (KASTEL)

Mutable Arguments

```
fn clamp(a: &mut ty!{ a1 : i32 | true => a2 | a2 <= b1 }, b: ty!{ b1: i32 }) {
  if *a > b { *a = b }
}
fn client() -> ty!{ v: () } {
  ...
  let max = 42;
  clamp(&mut x, max);
  x as ty!{ v : i32 | v < 43 };
}
```

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work
○○○○○○○○   ○○○○●○○○○   ○○○   ○○   ○○○

15/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                 Security and Dependability (KASTEL)

Mutable Arguments

```
fn clamp(
  a: &mut ty!{ a1 : i32 | true => a2 | a2 <= b1 },
  b: ty!{ b1: i32 }
) {
  // Γ₁ = ({a ↦ v₁, arg₀ ↦ a₁, b ↦ b₁},
  //           v₁ ≐ &arg₀ ∧ true ∧ true)
  if *a > b { *a = b }
  // Γ₂ = ({a ↦ v₁, arg₀ ↦ v₂, b ↦ b₁},
  //           v₂ ≤ b₁ ∧ v₁ ≐ &arg₀ ∧ true ∧ true)
}
```

Motivation ○○○○○○○○   Type System ○○○○○○●○○○   Soundness Justification ○○○   Related Work ○○   Conclusion / Future Work ○○○

16/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information Security and Dependability (KASTEL)

Mutable Arguments

```
fn clamp(
  a: &mut ty!{ a1 : i32 | true => a2 | a2 <= b1 },
  b: ty!{ b1: i32 }
) {
  // Γ₁ = ({a ↦ v₁, arg₀ ↦ a₁, b ↦ b₁},
  //              v₁ ≐ &arg₀ ∧ true ∧ true)
  if *a > b { *a = b }
  // Γ₂ = ({a ↦ v₁, arg₀ ↦ v₂, b ↦ b₁},
  //        v₂ ≤ b₁ ∧ v₁ ≐ &arg₀ ∧ true ∧ true)
}
```

- ty! $\{\alpha : \mathsf{b} \mid \varphi \Rightarrow \beta \mid \psi\}$
- Callee requires φ for reference destination α
- Callee ensures ψ for reference destination β
- Of course, multiple arguments possible

Motivation
○○○○○○○○

Type System
○○○○○●○○○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○○

16/27   26.10.2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Mutable Arguments

```
fn clamp(
  a: &mut ty!{ a1 : i32 | true => a2 | a2 <= b1 },
  b: ty!{ b1: i32 }
) {
  // Γ₁ = ({a ↦ v₁, arg₀ ↦ a₁, b ↦ b₁},
  //              v₁ ≐ &arg₀ ∧ true ∧ true)
  if *a > b { *a = b }
  // Γ₂ = ({a ↦ v₁, arg₀ ↦ v₂, b ↦ b₁},
  //              v₂ ≤ b₁ ∧ v₁ ≐ &arg₀ ∧ true ∧ true)
}
```

$\Gamma_1 = (\{a \mapsto v_1, arg_0 \mapsto a_1, b \mapsto b_1\},$
$\qquad\qquad v_1 \doteq \&arg_0 \wedge \text{true} \wedge \text{true})$

$\Gamma_2 = (\{a \mapsto v_1, arg_0 \mapsto v_2, b \mapsto b_1\},$
$\qquad\qquad v_2 \leq b_1 \wedge v_1 \doteq \&arg_0 \wedge \text{true} \wedge \text{true})$

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work
○○○○○○○○   ○○○○○○●○○○   ○○○   ○○   ○○○

16/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                     Security and Dependability (KASTEL)

Mutable Arguments

```
fn clamp(
  a: &mut ty!{ a1 : i32 | true => a2 | a2 <= b1 },
  b: ty!{ b1: i32 }
) {
```
$$\Gamma_1 = (\{a \mapsto v_1, arg_0 \mapsto a_1, b \mapsto b_1\},$$
$$v_1 \doteq \&arg_0 \wedge \text{true} \wedge \text{true})$$
```
  if *a > b { *a = b }
```
$$\Gamma_2 = (\{a \mapsto v_1, arg_0 \mapsto v_2, b \mapsto b_1\},$$
$$v_2 \le b_1 \wedge v_1 \doteq \&arg_0 \wedge \text{true} \wedge \text{true})$$
```
}
```

Motivation
○○○○○○○○

Type System
○○○○○○●○○○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○○

16/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Sub-Context

```
fn clamp(
  a: &mut ty!{ a1 : i32 | true => a2 | a2 <= b1 },
  b: ty!{ b1: i32 }
) {
  // Γ₁ = ({a ↦ v₁, arg₀ ↦ a₁, b ↦ b₁},
  //          v₁ ≐ &arg₀ ∧ true ∧ true)
  if *a > b { *a = b }
  // Γ₂ = ({a ↦ v₁, arg₀ ↦ v₂, b ↦ b₁},
  //          v₂ ≤ b₁ ∧ v₁ ≐ &arg₀ ∧ true ∧ true)
}
```

- still left: proof obligation from signature
  $a_2 \leq b_1$
- i.e. is Γ_2 a valid end-state?

Sub-Context

```
fn clamp(
  a: &mut ty!{ a1 : i32 | true => a2 | a2 <= b1 },
  b: ty!{ b1: i32 }
) {
  // Γ₁ = ({a ↦ v₁, arg₀ ↦ a₁, b ↦ b₁},
  //            v₁ ≐ &arg₀ ∧ true ∧ true)
  if *a > b { *a = b }
  // Γ₂ = ({a ↦ v₁, arg₀ ↦ v₂, b ↦ b₁},
  //            v₂ ≤ b₁ ∧ v₁ ≐ &arg₀ ∧ true ∧ true)
}
```

- still left: proof obligation from signature
  $a_2 \leq b_1$
- i.e. is Γ_2 a valid end-state?
- generalize notion of sub-types to context: sub-context

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work

16/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                                      Security and Dependability (KASTEL)

Sub-Context

```
fn clamp(
  a: &mut ty!{ a1 : i32 | true => a2 | a2 <= b1 },
  b: ty!{ b1: i32 }
) {
  // Γ1 = ({a ↦ v1, arg0 ↦ a1, b ↦ b1},
  //              v1 ≐ &arg0 ∧ true ∧ true)
  if *a > b { *a = b }
  // Γ2 = ({a ↦ v1, arg0 ↦ v2, b ↦ b1},
  //         v2 ≤ b1 ∧ v1 ≐ &arg0 ∧ true ∧ true)
}
```

- still left: proof obligation from signature $a_2 \leq b_1$
- i.e. is Γ_2 a valid end-state?
- generalize notion of sub-types to context: sub-context
- expected state:
  $\Gamma_e = (\{arg_0 \mapsto a_2, b \mapsto b_1\}, a_2 \leq b_1)$

Motivation
○○○○○○○○

Type System
○○○○○●○○○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○○

16/27 26.10.2022 Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Sub-Context

```
fn clamp(
  a: &mut ty!{ a1 : i32 | true => a2 | a2 <= b1 },
  b: ty!{ b1: i32 }
) {
  // Γ₁ = ({a ↦ v₁, arg₀ ↦ a₁, b ↦ b₁},
  //            v₁ ≐ &arg₀ ∧ true ∧ true)
  if *a > b { *a = b }
  // Γ₂ = ({a ↦ v₁, arg₀ ↦ v₂, b ↦ b₁},
  //          v₂ ≤ b₁ ∧ v₁ ≐ &arg₀ ∧ true ∧ true)
}
```

- still left: proof obligation from signature $a_2 \leq b_1$
- i.e. is Γ_2 a valid end-state?
- generalize notion of sub-types to context: sub-context
- expected state: $\Gamma_e = (\{arg_0 \mapsto a_2, b \mapsto b_1\}, a_2 \leq b_1)$
- show: $\Gamma_2 \preceq \Gamma_e$

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work

Sub-Context

$$\preceq\text{-C\tiny{TX}} \quad \frac{\vDash \Phi \to \Phi'[\mu'(x) \rhd \mu(x) \mid x \in \text{dom}(\mu')] \qquad \text{dom}(\mu') \subseteq \text{dom}(\mu)}{(\mu, \Phi) \preceq (\mu', \Phi')}$$

- still left: proof obligation from signature
  $a_2 \leq b_1$
- i.e. is Γ_2 a valid end-state?
- generalize notion of sub-types to context: sub-context
- expected state:
  $\Gamma_e = (\{arg_0 \mapsto a_2, b \mapsto b_1\}, a_2 \leq b_1)$
- show: $\Gamma_2 \preceq \Gamma_e$

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work
00000000   00000●000   000   00   000

16/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Mutable Calls

```
fn client(...) -> ty!{ v: () } {
  ...
  let m = 42;
  // Γ₁ = ({x ↦ v₁, m ↦ v₂}, ... ∧ v₂ ≐ 42)
  clamp(&mut x, m);
  // Γ₂ = ({x ↦ v₃, m ↦ v₂}, ... ∧ v₂ ≐ 42 ∧ v₃ ≤ 5)
  x as ty!{ v : i32 | v < 43 };
}
```

- append predicates from callee to context
- update association of logic variables

```rust
#![allow(dead_code)]
use runtime_library::*;

fn clamp(
  a: &mut ty!{ a1 : i32 | true ⇒ a2 | a2 ⩽ b1 },
  b: ty!{ b1: i32 }
) → ty!{ v: () } {
  if *a > b {
    *a = b as ty!{ r | (r ⩽ b1) }; ()
  } else {};
  ()
}

fn client() → ty!{ v: () } {
  let mut x: i32 = 1337; let max: i32 = 42;
  clamp(a: &mut x, b: max);
  x as ty!{ v : i32 | v < 43 };
}
```

Motivation
○○○○○○○○

Type System
○○○○○○○●○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○○

```rust
#![allow(dead_code)]
use runtime_library::*;

fn clamp(
    a: &mut ty!{ a1 : i32 | true => a2 | a2 <= b1 },
    b: ty!{ b1 : i32 }
) -> ty!{ v : () } {
    if *a > b {
        *a = b as ty!{ r | (r <= b1) }; ()
    } else {};
    ()
}

fn client() -> ty!{ v : () } {
    let mut x: i32 = 1337; let max: i32 = 42;
    clamp(a: &mut x, b: max);
    x as ty!{ v : i32 | v < 41 };
}
```

Subtyping judgement failed:

Motivation
○○○○○○○○

Type System
○○○○○○○●○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○○

18/27   26.10.2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

SMT Request

```
; checking is_sub_context ...
(declare-datatypes () ((Unit unit)))
(declare-const |_0| Int)
(declare-const |r| Int)
(declare-const |a1| Int)
(declare-const |b1| Int)
(declare-const |a2| Int)

; ty!{ r : i32 | (r <= b1) }
(assert (<= |r| |b1|))

; ty!{ a1 : &mut i32 | true }
(assert true)

;    ty!{ b1 : i32 | true }
(assert true)
```

```
; SuperCtx:
(assert (not (and
        (<= |r| |b1|)
        true
    )))

; checking: RContext {
;     a : ty!{ _0 : &mut i32 | _0 == & arg (0usize) }
;     <dangling> : ty!{ a1 : &mut i32 | true }
;     b : ty!{ b1 : i32 | true }
;     <argument 0> : ty!{ r : i32 | (r <= b1) }
; }
;  <: RContext {
;     <argument 0> : ty!{ a2 : &mut i32 | a2 <= b1 }
;     b : ty!{ b1 : i32 | true }
; }
(check-sat)
```

Motivation
○○○○○○○○

Type System
○○○○○○○○●

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○○

19/27   26.10.2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Soundness

Progress

If $\Gamma \vdash s_1$, $\sigma : \Gamma \Rightarrow \Gamma_2$ and $s_1 \neq$ unit, then there is a s_2 and σ_2 with $\langle s_1 \mid \sigma_1 \rangle \rightsquigarrow \langle s_2 \mid \sigma_2 \rangle$.

Corten strictly refines the base language, therefore progress depends on base type system.

Preservation

If $\Gamma \vdash s \Rightarrow \Gamma_2$, $\sigma : \Gamma$ and $\langle s \mid \sigma \rangle \rightsquigarrow \langle s_1 \mid \sigma_1 \rangle$, then there is a Γ_1 with $\Gamma_1 \vdash s_1 \Rightarrow \Gamma_2$ and $\sigma_2 : \Gamma_2$

Stronger property than base language preservation: Show that refined types are preserved

Partial, Mechanized Proof in Lean 4

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work

20/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information Security and Dependability (KASTEL)

State Conformance

State Conformance $\sigma : \Gamma$

A state σ is conformant with respect to a typing context $\Gamma = (\mu, \Phi)$ (written as $\sigma : \Gamma$), iff:

$$\Phi[\mu(x) \triangleright \llbracket \sigma(x) \rrbracket \mid x \in \text{dom}(\mu)] \text{ is satisfiable}$$

I.e. a conformant type context does not contradict the execution state.

Examples:

- If $\sigma : (\emptyset, \Phi)$ then Φ is satisfiable
- If $\sigma : (\mu, \Phi_1 \wedge \Phi_2)$ then $\sigma : (\mu, \Phi_1)$ and $\sigma : (\mu, \Phi_1)$.
- If $\sigma : (\mu, \Phi)$ and $\text{FV}(\Phi) \subseteq \text{dom}(\mu)$, then $\vDash \Phi[\mu(x) \triangleright \llbracket \sigma(x) \rrbracket \mid x \in \text{dom}(\mu)]$

Motivation
○○○○○○○○

Type System
○○○○○○○○○

Soundness Justification
○●○

Related Work
○○

Conclusion / Future Work
○○○

21/27 26.10.2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Intermediate Steps

Conformance of Symbolic Execution

If $\sigma : \Gamma$, $\Gamma \vdash \alpha$ fresh then $\sigma[x \mapsto [\![e]\!]\sigma] : \Gamma[x \mapsto \alpha], (\alpha \simeq [\![e]\!]\Gamma)$

where $(\alpha \simeq [\![e]\!]\Gamma)$ is the symbolic execution of e equated with α in context Γ

Reference Predicates are Conservative

If $\sigma : \Gamma$ and $\Gamma \vdash *x \in \{y_1, \ldots, y_n\}$ then $[\![\sigma(x)]\!] = \& y_i$ for some $i \in 1, \ldots, n$

Rare case where conservative typing requires

Sub-Context Relation is Conservative

If $\Gamma \preceq \Gamma'$ and $\sigma : \Gamma$ then $\sigma : \Gamma'$

Motivation   Type System   Soundness Justification   Related Work   Conclusion / Future Work

22/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
                                                                                     Security and Dependability (KASTEL)

Related Work

Refinement Types and Mutability

- Rondon et al. [RKJ10], Bakst and Jhala [BJ16]: Refinement Types for C subset. Lack of guarantees requires ad-hoc mechanisms to control aliasing
- Lanzinger [Lan21]: Property Types in Java (only immutable). Bachmeier [Bac22]: Extension using Ownership System
- Toman et al. [Tom+20] (ConSORT): Fractional Ownership, strong and weak updates

Rust verification

- Ullrich [Ull16]: Translation to Lean; linear mutation chain. Denis et al [DJM21] similar, but to Why3
- Astrauskas et al. [Ast+19] (Prusti): heavy-weight verification, translation to separation logic (Viper)
- Matsushita et al. [MTK20] (RustHorn): constrained Horn clauses

Motivation	Type System	Soundness Justification	Related Work	Conclusion / Future Work
○○○○○○○○	○○○○○○○○○	○○○	●○	○○○

Related Work: Flux – Refinement Types for Rust

- MIR vs. HIR
- specification in comments vs. embedding in types
- context inclusions vs. sub context
- distinction strong and weak references vs. dynamic choice by typ checking rules
- explicit introduction of logic variables vs. ad-hoc
- formalization based on RustBelt vs. formalization based on own language
- missing in Corten: records & inference
- otherwise: similar capabilities

```
// Flux
//@ ensures *self: i32<n+1>;
fn increment(&strg v : i32<n>) -> ()
//@ requires n > 0
//@ ensures *self: i32<n-1>;
fn decrement(&strg v : i32<n>) -> ()
// Corten
fn increment(n: &mut ty!{
  n1: Nat => n1 | n1 == n1+1 }
) -> ();
fn decrement(n: &mut ty!{
  v1: Nat | v1 > 0 => v2 | v2 == v1-1 }
) -> ();
```

Motivation   Type System   Soundness Justification   **Related Work**   Conclusion / Future Work

24/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information Security and Dependability (KASTEL)

Future Work

- Records & ADTs
  - More Syntax, Nested Structures
  - Variant Distinction
- Predicate Generics (Abstract Predicates)
  - Uninterpreted Functions in Types
  - Syntactic Embedding?
- Concurrency using Predicate Generics?
  - Use Predicate Generics
  - Predicate describes Contract for Mutation
  - Interesting, because unusual guarantees in Rust

Motivation

Type System

Soundness Justification

Related Work

Conclusion / Future Work

25/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Conclusion

- Refinement Type System for Rust with Mutability
  - Decidable, Automatic
  - Complex Mutation Patterns
  - ...
- Minimal Interface
- Soundness Justification
- Practical Usability
  - Source Locations
  - Counter Example
  - IDE Integration

Motivation
○○○○○○○○

Type System
○○○○○○○○○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○●○

26/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Conclusion

- Refinement Type System for Rust with Mutability
    - Decidable, Automatic
    - Complex Mutation Patterns
    - ...
- Minimal Interface
- Soundness Justification
- Practical Usability
    - Source Locations
    - Counter Example
    - IDE Integration

More Information:

- Implementation, Thesis, Mechanized Proof, Evaluation:
  https://gitlab.com/csicar/liquidrust
- Empirical Analysis:
  https://gitlab.com/csicar/crates-analysis

Motivation
○○○○○○○○

Type System
○○○○○○○○○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○●○

26/27   26.10.2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Motivation
○○○○○○○○

Type System
○○○○○○○○○

Soundness Justification
○○○

Related Work
○○

Conclusion / Future Work
○○●

27/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Literatur I

[1]   Vytautas Astrauskas u. a. „Leveraging rust types for modular specification and verification". In: *Proceedings of the ACM on Programming Languages* 3 (OOPSLA 10. Okt. 2019), S. 1–30. ISSN: 2475-1421. DOI: 10.1145/3360573. URL: https://dl.acm.org/doi/10.1145/3360573 (besucht am 23. 02. 2022).

[2]   Joshua Bachmeier. *Property Types for Mutable Data Structures in Java*. 2022. DOI: 10.5445/IR/1000150318. URL: https://publikationen.bibliothek.kit.edu/1000150318 (besucht am 03. 10. 2022).

[3]   Alexander Bakst und Ranjit Jhala. „Predicate Abstraction for Linked Data Structures". In: *Verification, Model Checking, and Abstract Interpretation*. Hrsg. von Barbara Jobstmann und K. Rustan M. Leino. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2016, S. 65–84. ISBN: 978-3-662-49122-5. DOI: 10.1007/978-3-662-49122-5_3.

[4] Xavier Denis, Jacques-Henri Jourdan und Claude Marché. „The Creusot Environment for the Deductive Verification of Rust Programs". Diss. Inria Saclay-Île de France, 2021.

[5] Florian Lanzinger. „Property Types in Java: Combining Type Systems and Deductive Verification". Master Thesis. Karlsruher Institut für Technologie, Feb. 2021.

[6] Yusuke Matsushita, Takeshi Tsukada und Naoki Kobayashi. „RustHorn: CHC-based verification for Rust programs". In: *European Symposium on Programming*. Springer, Cham, 2020, S. 484–514.

[7] Patrick M. Rondon, Ming Kawaguci und Ranjit Jhala. „Liquid types". In: *Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation - PLDI '08*. the 2008 ACM SIGPLAN conference. Tucson, AZ, USA: ACM Press, 2008, S. 159. ISBN: 978-1-59593-860-2. DOI: 10.1145/1375581.1375602. URL: http://portal.acm.org/citation.cfm?doid=1375581.1375602 (besucht am 30. 06. 2022).

[8] Patrick Maxim Rondon, Ming Kawaguchi und Ranjit Jhala. „Low-level liquid types". In: *Proceedings of the 37th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. POPL '10. New York, NY, USA: Association for Computing Machinery, 17. Jan. 2010, S. 131–144. ISBN: 978-1-60558-479-9. DOI: 10.1145/1706299.1706316. URL: https://doi.org/10.1145/1706299.1706316 (besucht am 16.09.2022).

[9] John Toman u. a. „ConSORT: Context- and Flow-Sensitive Ownership Refinement Types for Imperative Programs". In: *Programming Languages and Systems*. Hrsg. von Peter Müller. Cham: Springer International Publishing, 2020, S. 684–714. ISBN: 978-3-030-44914-8. DOI: 10.1007/978-3-030-44914-8_25.

[10] Sebastian Ullrich. „Simple Verification of Rust Programs via Functional Purification". In: (6. Dez. 2016), S. 65.

Empirical Use-Case Analysis

- public open-source code (crates.io)
- about 64 million lines of Rust code
- syntactical analysis
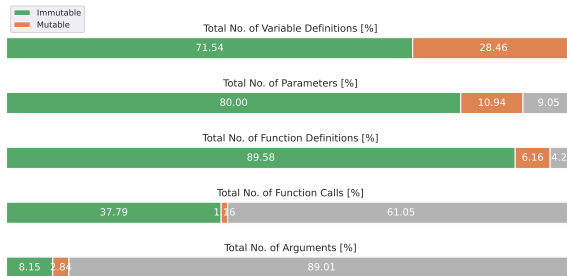
let $\Gamma_2 = \Gamma[i \mapsto v_1], v > 0$ and $\tau = \{v : \text{i32} \mid v > 0\}$

$$\text{SEQ} \cfrac{\text{DECL} \cfrac{\text{INTRO-SUB} \cfrac{\Gamma_1 \vdash \ldots : \tau' \qquad \Gamma_1 \vdash \tau' \preceq \tau}{\Gamma_1 \vdash \ldots \text{ as } \tau : \tau}}{\Gamma_1 \vdash \text{let } i = \ldots \text{ as } \tau \Rightarrow \Gamma_2} \qquad \text{ASS} \cfrac{\text{BINOP} \cfrac{\Gamma_1 \vdash v_2 \text{ fresh}}{\Gamma_1 \vdash i - 1 : \{v_2 : \text{i32} \mid v_2 \doteq v - 1\}}}{\Gamma_2 \vdash i = i - 1 \Rightarrow \Gamma[i \mapsto v_2], v > 0, v_2 \doteq v - 1}}{\Gamma_1 \vdash \text{let } i = \ldots \text{ as } \tau; \ i = i - 1 \Rightarrow \Gamma[i \mapsto v_2], v > 0, v_2 \doteq v - 1}$$

Empirical Analysis
○○○○○○○

Zweiter Abschnitt
○●○○

Farben
○

32/27 26.10.2022 Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Expression Typing $\Gamma \vdash e : \tau$

$$\text{LIT } \frac{\Gamma \vdash \alpha \text{ fresh}}{\Gamma \vdash v : \{\alpha : b \mid \alpha \simeq [\![v]\!]\Gamma\}} \qquad \text{BINOP } \frac{\Gamma \vdash \alpha \text{ fresh}}{\Gamma \vdash x_1 \odot x_2 : \{\alpha : b \mid \alpha \simeq [\![x_1 \odot x_2]\!]\Gamma\}}$$

$$\text{VAR } \frac{\Gamma \vdash \alpha \text{ fresh}}{\Gamma \vdash x : \{\alpha : b \mid \alpha \simeq [\![x]\!]\Gamma\}} \qquad \text{INTRO-SUB } \frac{\Gamma \vdash e : \tau \qquad \Gamma \vdash \tau \preceq \tau'}{\Gamma \vdash e \text{ as } \tau' : \tau'}$$

Statement Type Checking $\Gamma \vdash s \Rightarrow \Gamma'$

$$\text{IF } \frac{\Gamma, \Gamma(x) \doteq \text{true} \vdash s_t \Rightarrow \Gamma' \qquad \Gamma, \Gamma(x) \doteq \text{false} \vdash s_e \Rightarrow \Gamma'}{\Gamma \vdash \text{if } x \text{ then } s_t \text{ else } s_e \Rightarrow \Gamma'}$$

$$\text{SEQ } \frac{\Gamma \vdash s_1 \Rightarrow \Gamma' \qquad \Gamma' \vdash s_2 \Rightarrow \Gamma''}{\Gamma \vdash s_1; s_2 \Rightarrow \Gamma''}$$

$$\text{DECL } \frac{\Gamma \vdash e : \{\beta : b \mid \varphi\}}{\Gamma \vdash \text{let } x = e \Rightarrow \Gamma[x \mapsto \beta], \varphi} \qquad \text{ASSIGN } \frac{\Gamma \vdash e : \{\beta : b \mid \varphi\}}{\Gamma \vdash x = e \Rightarrow \Gamma[x \mapsto \beta], \varphi}$$

Empirical Analysis
○○○○○○○

Zweiter Abschnitt
○○●○

Farben
○

33/27   26.10.2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Expression Typing $\Gamma \vdash e : \tau$

$$\text{REF} \; \frac{\Gamma \vdash \alpha \; \text{fresh}}{\Gamma \vdash \&x : \{\alpha : \&b \mid \alpha \simeq [\![\&x]\!]\Gamma\}}$$

$$\text{VAR-DEREF} \; \frac{\Gamma \vdash x \in \{\&y\} \quad \Gamma \vdash y : \tau}{\Gamma \vdash *x : \tau}$$

Statement Type Checking $\Gamma \vdash s \Rightarrow \Gamma'$

$$\text{ASSIGN-STRONG} \; \frac{\Gamma(z) = \beta \quad \Gamma \vdash x \in \{\&y\} \quad \Gamma \vdash \gamma \; \text{fresh}}{\Gamma \vdash *x = z \Rightarrow \Gamma[y \mapsto \gamma], \gamma \doteq \beta}$$

Empirical Analysis
○○○○○○○

Zweiter Abschnitt
○○○●

Farben
○

34/27   26.10.2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Expression Typing $\Gamma \vdash e : \tau$

$$\text{REF} \quad \frac{\Gamma \vdash \alpha \text{ fresh}}{\Gamma \vdash \&x : \{\alpha : \&b \mid \alpha \simeq \llbracket \&x \rrbracket \Gamma\}}$$

$$\text{VAR-DEREF} \quad \frac{\Gamma \vdash x \in \{\&y\} \qquad \Gamma \vdash y : \tau}{\Gamma \vdash *x : \tau}$$

Statement Type Checking $\Gamma \vdash s \Rightarrow \Gamma'$

$$\text{ASSIGN-STRONG} \quad \frac{\Gamma(z) = \beta \qquad \Gamma \vdash x \in \{\&y\} \qquad \Gamma \vdash \gamma \text{ fresh}}{\Gamma \vdash *x = z \Rightarrow \Gamma[y \mapsto \gamma], \gamma \doteq \beta}$$

$$\text{ASSIGN-WEAK} \quad \frac{\begin{array}{c} \Gamma \vdash e : \tau \qquad \Gamma \vdash x \in \{\&y_1, \ldots, \&y_n\} \\ \hline \Gamma \vdash y_i : \{\beta_i : b_i \mid \varphi_i\} \qquad \Gamma \vdash \tau \preceq \{\beta_i : b_i \mid \varphi_i\} \end{array}}{\Gamma \vdash *x = e \Rightarrow \Gamma}$$

Empirical Analysis

Zweiter Abschnitt

Farben

34/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Predicate Generics & Concurrecy

```
struct Mutex<P, T> = ...;

impl Mutex<P, T> {
  fn lock() -> MutexLock<P, T> {
    ...
  }
}

struct MutexLock<P, T> = ...;

impl MutexLock<P, T> {
  fn drop(self : ty!{ v : T | P v }) {
    ...
  }
}
```

Empirical Analysis
ooooooo

Zweiter Abschnitt
oooo

Farben
o

35/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Blöcke
in den KIT-Farben

Greenblock		Blueblock		Redblock	
Standard (`block`)		= `exampleblock`		= `alertblock`	

Brownblock	Purpleblock	Cyanblock

Yellowblock	Lightgreenblock	Orangeblock

Grayblock	**Contentblock**
	(farblos)

Empirical Analysis
○○○○○○○

Zweiter Abschnitt
○○○○

Farben
○

36/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

Auflistungen

Text

- Auflistung
  Umbruch
- Auflistung
  - Auflistung
  - Auflistung

Empirical Analysis
0000000

Zweiter Abschnitt
0000

Farben
0

37/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

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

Empirical Analysis

Zweiter Abschnitt

Farben

38/27   26. 10. 2022   Carsten Csiky: Rust & Refinement Types

Department of Informatics – Institute of Information
Security and Dependability (KASTEL)

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

kit-green100	kit-green90	kit-green80	kit-green70	kit-green60	kit-green50	kit-green40	kit-green30	kit-green25	kit-green20	kit-green15	kit-green10	kit-green5
kit-blue100	kit-blue90	kit-blue80	kit-blue70	kit-blue60	kit-blue50	kit-blue40	kit-blue30	kit-blue25	kit-blue20	kit-blue15	kit-blue10	kit-blue5
kit-red100	kit-red90	kit-red80	kit-red70	kit-red60	kit-red50	kit-red40	kit-red30	kit-red25	kit-red20	kit-red15	kit-red10	kit-red5
kit-gray100	kit-gray90	kit-gray80	kit-gray70	kit-gray60	kit-gray50	kit-gray40	kit-gray30	kit-gray25	kit-gray20	kit-gray15	kit-gray10	kit-gray5
kit-orange100	kit-orange90	kit-orange80	kit-orange70	kit-orange60	kit-orange50	kit-orange40	kit-orange30	kit-orange25	kit-orange20	kit-orange15	kit-orange10	kit-orange5
kit-lightgreen100	kit-lightgreen90	kit-lightgreen80	kit-lightgreen70	kit-lightgreen60	kit-lightgreen50	kit-lightgreen40	kit-lightgreen30	kit-lightgreen25	kit-lightgreen20	kit-lightgreen15		
kit-lightgreen10	kit-lightgreen5											
kit-brown100	kit-brown90	kit-brown80	kit-brown70	kit-brown60	kit-brown50	kit-brown40	kit-brown30	kit-brown25	kit-brown20	kit-brown15	kit-brown10	kit-brown5
kit-purple100	kit-purple90	kit-purple80	kit-purple70	kit-purple60	kit-purple50	kit-purple40	kit-purple30	kit-purple25	kit-purple20	kit-purple15	kit-purple10	kit-purple5
kit-cyan100	kit-cyan90	kit-cyan80	kit-cyan70	kit-cyan60	kit-cyan50	kit-cyan40	kit-cyan30	kit-cyan25	kit-cyan20	kit-cyan15	kit-cyan10	kit-cyan5

Empirical Analysis
○○○○○○○

Zweiter Abschnitt
○○○○

Farben
○

42/27   26.10.2022   Carsten Csiky: Rust & Refinement Types   Department of Informatics – Institute of Information
Security and Dependability (KASTEL)