RustBelt

Securing the Foundations of the Rust Programming Language

Luko van der Maas
Ceel Pierik

Radboud University Nijmegen

January 12, 2021

Contents

Introduction

Recap of important parts of Rust

λ_{Rust}

The goal of Rust

Control vs. safety

The goal of Rust

Control vs. safety

Rust: having the cake and eating it too.

- no garbage collector, zero-cost abstractions
- type-safe, memory-safe
- statically ruling out data races

The goal of Rust

Control vs. safety

Rust: having the cake and eating it too.

- no garbage collector, zero-cost abstractions
- type-safe, memory-safe
- statically ruling out data races

Strong claims but: can we prove it?

Ownership

Ownership: stopping uncontrolled aliasing and mutating

- statically enforced
- multiple aliases, one owner
- controlled shared ownership

However...

Ownership

Ownership: stopping uncontrolled aliasing and mutating

- statically enforced
- multiple aliases, one owner
- controlled shared ownership

However... guaranteeing safety of advanced low-level programming is difficult, especially if we want to keep automatic type checking. This has resulted in:

- too restrictive type systems
- too expressive type systems

Rust's solution

The extensible approach:

- basis: shared ownership => no direct mutation
- use `unsafe` operations in libraries
- 'encapsulate' these libraries: safe API

Result: Rust's expressive power increases through externally safe libraries by loosening its ownership discipline

Difficulties

1. Soundness bugs have been found.
2. "Progress and preservation" cannot be used to prove safety.

RustBelt

Enter **RustBelt**: a semantic model of λ_{Rust}.

RustBelt

Enter **RustBelt**: a semantic model of λ_{Rust}.

Three-part proof of λ_{Rust}safety:

1. Verify the *fundamental theorem of logical relations*.
2. Verify *adequacy*.
3. For any library, verify that its implementation satisfies the predicate associated with the semantic interpretation of its interface.

Result: extensible soundness proof for λ_{Rust}.

Contents

Introduction

Recap of important parts of Rust

λ_{Rust}

Ownership and borrows

Three types of objects

```
1  let(snd , rcv) = channel ();
2  join(move || {
3          let mut v = Vec::new(); v.push(0); // v: Vec <i32>
4          snd.send(v); // send: fn(&mut Sender<Vec<i32>>, Vec<i32>)
5          //  Cannot access v: v.push (1)  rejected
6          //  Can access snd
7      },
8      move || {
9          let v = rcv.recv().unwrap(); // v: Vec <i32>
10         println!("Received: {:?}", v);
11         // ends up as: fn write(&mut self, buf: &[u8])
12 });
```

Ownership and borrows

Three types of objects

- Exclusive ownership

```
1   let(snd , rcv) = channel ();
2   join(move || {
3           let mut v = Vec::new(); v.push(0); // v: Vec <i32>
4           snd.send(v); // send: fn(&mut Sender<Vec<i32>>, Vec<i32>)
5           //  Cannot access v: v.push (1)  rejected
6           //  Can access snd
7       },
8       move || {
9           let v = rcv.recv().unwrap(); // v: Vec <i32>
10          println!("Received: {:?}", v);
11          // ends up as: fn write(&mut self, buf: &[u8])
12  });
```

Ownership and borrows

Three types of objects

- Exclusive ownership
- Mutable references

```
1   let(snd , rcv) = channel ();
2   join(move || {
3           let mut v = Vec::new(); v.push(0); // v: Vec <i32>
4           snd.send(v); // send: fn(&mut Sender<Vec<i32>>, Vec<i32>)
5           //  Cannot access v: v.push (1)  rejected
6           //  Can access snd
7       },
8       move || {
9           let v = rcv.recv().unwrap(); // v: Vec <i32>
10          println!("Received: {:?}", v);
11          // ends up as: fn write(&mut self, buf: &[u8])
12  });
```

Ownership and borrows

Three types of objects

- Exclusive ownership
- Mutable references
- Shared references

```
1  let(snd , rcv) = channel ();
2  join(move || {
3          let mut v = Vec::new(); v.push(0); // v: Vec <i32>
4          snd.send(v); // send: fn(&mut Sender<Vec<i32>>, Vec<i32>)
5          //  Cannot access v: v.push (1)  rejected
6          //  Can access snd
7      },
8      move || {
9          let v = rcv.recv().unwrap(); // v: Vec <i32>
10         println!("Received: {:?}", v);
11         // ends up as: fn write(&mut self, buf: &[u8])
12 });
```

Lifetimes

```
1  fn example(v: &/* 'a */mut Vec<i32>) {
2      v.push(21);
3
4
5      v[0] = 23;
6
7
8      v.push(42);
9      println!("{:?}", v); // Prints [23, ..., 42]
10 }
```

Lifetimes

```
1  fn example(v: &/* 'a */mut Vec<i32>) {
2      v.push(21);
3      {/* 'b */
4        // index_mut: for<'b> fn(&'b mut Vec<i32>, usize) -> &'b mut i32
5        let mut head : &/* 'b */mut i32 = v.index_mut(0);
6        // Cannot access v: v.push(2) rejected
7        *head = 23; }
8      v.push(42);
9      println!("{:?}", v); // Prints [23, ..., 42]
10 }
```

Lifetimes

```
1   fn example(v: &/* 'a */mut Vec<i32>) {
2       v.push(21);
3       {/* 'b */
4         // index_mut: for<'b> fn(&'b mut Vec<i32>, usize) -> &'b mut i32
5         let mut head : &/* 'b */mut i32 = v.index_mut(0);
6         // Cannot access v: v.push(2) rejected
7         *head = 23; }
8       v.push(42);
9       println!("{:?}", v); // Prints [23, ..., 42]
10  }
```

- head is only valid inside the code block in which it was defined as that is the lifetime it got

Lifetimes

```
1  fn example(v: &/* 'a */mut Vec<i32>) {
2      v.push(21);
3      {/* 'b */
4        // index_mut: for<'b> fn(&'b mut Vec<i32>, usize) -> &'b mut i32
5        let mut head : &/* 'b */mut i32 = v.index_mut(0);
6        // Cannot access v: v.push(2) rejected
7        *head = 23; }
8      v.push(42);
9      println!("{:?}", v); // Prints [23, ..., 42]
10 }
```

- head is only valid inside the code block in which it was defined as that is the lifetime it got
- v is not valid during that time as there is an alive borrow of it which is kept alive by giving it the lifetime of v

Lifetimes

```
1  fn example(v: &/* 'a */mut Vec<i32>) {
2      v.push(21);
3      {/* 'b */
4        // index_mut: for<'b> fn(&'b mut Vec<i32>, usize) -> &'b mut i32
5        let mut head : &/* 'b */mut i32 = v.index_mut(0);
6        // Cannot access v: v.push(2) rejected
7        *head = 23; }
8      v.push(42);
9      println!("{:?}", v); // Prints [23, ..., 42]
10 }
```

- head is only valid inside the code block in which it was defined as that is the lifetime it got
- v is not valid during that time as there is an alive borrow of it which is kept alive by giving it the lifetime of v
- v is valid again after the code block as the borrow dies because it got the lifetime of v

Contents

Introduction

Recap of important parts of Rust

λ_{Rust}

Key features

λ_{Rust}: language to formally model important Rust features such as borrowing, lifetimes, and lifetime inclusion.

No traits, no relaxed memory model.

Some key features of λ_{Rust}:

- **continuation-passing style** to represent complex control-flow.
- **concise type system** by having individual instructions represent single operations.
- **memory model** that supports pointer arithmetic and ensures that programs with data races or illegal memory accesses can reach a stuck state in the operational semantics.

Syntax

$$
\begin{aligned}
\mathit{Path} \ni p &::= x \mid p.n \\
\mathit{Val} \ni v &::= \textbf{false} \mid \textbf{true} \mid z \mid l \mid \textbf{funrec}\ f(\overline{x})\ \textbf{ret}\ k := F \\
\mathit{Instr} \ni I &::= v \mid p \mid p_1 + p_2 \mid p_1 - p_2 \mid p_1 \leq p_2 \mid p_1 = p_2 \\
&\quad \mid \textbf{new}(n) \mid \textbf{delete}(n, p) \mid *p \mid p_1 := p_2 \mid p_1 :=_n *p_2 \\
&\quad \mid p \overset{\text{inj } i}{:==} () \mid p_1 \overset{\text{inj } i}{:==} p_2 \mid p_1 \overset{\text{inj } i}{:==}_n *p_2 \mid \ldots \\
\mathit{FuncBody} \ni F &::= \textbf{let}\ x = I\ \textbf{in}\ F \mid \textbf{letcont}\ k(\overline{x}) := F_1\ \textbf{in}\ F_2 \\
&\quad \mid \textbf{newlft}; F \mid \textbf{endlft}; F \mid \textbf{if}\ p\ \textbf{then}\ F_1\ \textbf{else}\ F_2 \mid \\
&\quad \textbf{case}\ *p\ \textbf{of}\ \overline{F} \mid \textbf{jump}\ k(\overline{x}) \mid \textbf{call}\ f(\overline{x})\ \textbf{ret}\ k
\end{aligned}
$$

Simple rust program

```
1      fn option_as_mut <'a >
2              (x: &'a mut Option<i32>) ->
3              Option<&'a mut i32> {
4          match *x {
5              None => None,
6              Some(ref mut t) => Some(t)
7          }
8      }
```

```
funrec option_as_mut(x) ret ret :=
    let r = new(2) in
    letcont k() := delete(1,x); jump ret(r) in
    let y = *x in case *y of
        r :=inj 0 (); jump k()
        r :=inj 1 y.1; jump k()
```

Loops rust

```
1  fn half(n: i32) -> Option<i32> {
2      let mut i = 0;
3      loop {
4          if i+i == n {
5              break Some(i);
6          } else if i == n {
7              break None;
8          } else {
9              i += 1;
10         }
11     }
12 }
```

Loops

```
funrec half(n) ret ret :=
  let r = new(2) in
  let i = new(1) in i := 0;
  letcont k := delete(1,n); delete(1,i);
    jump ret(r) in
  letcont loop() :=
    let i^pl = *i in let i^pr = *i in let n' = *n in
    let a = i^pl + i^pr in let c = a = n' in
    if c then let i' = *i in r :==(inj 1) i'; jump k()
    else let i'' = *i in let n'' = *n in
        let c' = i'' = n'' in
      if c' then r :==(inj 0) (); jump k()
      else let i''' = *i in let one = 1 in
        let i_new = i''' + one in i := i_new;
        jump loop()
  in jump loop()
```

```
1   fn half(n: i32) -> Option<i32>
2     let mut i = 0;
3     loop {
4       if i+i == n {
5         break Some(i);
6       } else if i == n {
7         break None;
8       } else {
9         i += 1;
10      }
11    }
12  }
```

Type system

$$\mathit{Lft} \ni \kappa ::= \alpha \mid \textbf{static}$$

$$\mathit{Mod} \ni \mu ::= \textbf{mut} \mid \textbf{shr}$$

$$\mathit{Sort} \ni \sigma ::= \textbf{val} \mid \textbf{lft} \mid \textbf{type}$$

$$\mathit{Type} \ni \tau ::= T \mid \textbf{bool} \mid \textbf{int} \mid \textbf{own}_n \tau \mid \&^{\kappa}_{\mu} \tau \mid \lightning_n \mid \Pi\overline{\tau} \mid \Sigma\overline{\tau} \mid$$
$$\forall \overline{\alpha}.\textbf{fn}(F : \mathbf{E}; \overline{\tau}) \to \tau \mid \mu T.\tau$$

$$\Gamma ::= \emptyset \mid \Gamma, X : \sigma$$

$$\mathbf{E} ::= \emptyset \mid \mathbf{E}, \kappa \sqsubseteq_e \kappa'$$

$$\mathbf{L} ::= \emptyset \mid \mathbf{L}, \kappa \sqsubseteq_l \overline{\kappa}$$

$$\mathbf{K} ::= \emptyset \mid \mathbf{K}, k \lhd \textbf{cont}(\mathbf{L}; \overline{x}.\mathbf{T})$$

$$\mathbf{T} ::= \emptyset \mid \mathbf{T}, p \lhd \tau \mid \mathbf{T}, p \lhd^{\dagger\kappa} \tau$$

Function body judgment: $\Gamma \mid \mathbf{E}; \mathbf{L} \mid \mathbf{K}; \mathbf{T} \vdash F$

Instruction judgment: $\Gamma \mid \mathbf{E}; \mathbf{L} \mid \mathbf{T}_1 \vdash I \dashv x.\mathbf{T}_2$

Type checking example

```
funrec option_as_mut(x) ret ret :=
```

$$\Gamma := x : \textbf{val}, ret : \textbf{val}, \alpha : \textbf{lft}, F : \textbf{lft}$$

$$\mathbf{E} := F \sqsubseteq_e \alpha$$

$$\mathbf{L} := F \sqsubseteq_l []$$

$$\mathbf{K} := ret \lhd \textbf{cont}\,(F \sqsubseteq_l [];\, r.r \lhd \textbf{own}\,(\textbf{()} + \&^{\alpha}_{\textbf{mut}}\textbf{int}))$$

$$\mathbf{T} := x \lhd \textbf{own}\,\&^{\alpha}_{\textbf{mut}}(\textbf{()} + \textbf{int})$$

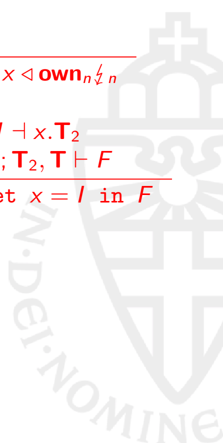
Type checking example

```
funrec option_as_mut(x) ret ret :=
```

$\{\mathbf{K} : ret \lhd \mathbf{cont}\,(F \sqsubseteq_l [\,]; r.r \lhd \mathbf{own}((\,)+\&^{\alpha}_{\mathbf{mut}}\mathbf{int})); \mathbf{T} : x \lhd \mathbf{own}\&^{\alpha}_{\mathbf{mut}}((\,)+\mathbf{int})\}$

```
  let r = new(2) in

  letcont k() := delete(1,x); jump ret(r) in

  let y = *x in

  case *y of
    r :=inj 0 (); jump k()

    r :=inj 1 y.1;

    jump k()
```

Type checking example



$\Gamma := x : \textbf{val}, ret : \textbf{val}, \alpha : \textbf{lft}, F : \textbf{lft}$

$\textbf{E} := F \sqsubseteq_e \alpha$

$\textbf{L} := F \sqsubseteq_l []$

$\textbf{K} := ret \triangleleft \textbf{cont}\,(F \sqsubseteq_l []; r.r \triangleleft \textbf{own}\,(() + \&^{\alpha}_{\textbf{mut}}\textbf{int}))$

$\textbf{T} := x \triangleleft \textbf{own}\,\&^{\alpha}_{\textbf{mut}}(() + \textbf{int})$

$$\text{S-NEW}\;\frac{}{\textbf{E}; \textbf{L} \mid \emptyset \vdash \texttt{new}(n) \dashv x.x \triangleleft \textbf{own}_n ⯑_n}$$

$$\text{F-LET}\;\frac{\Gamma \mid \textbf{E}; \textbf{L} \mid \textbf{T}_1 \vdash I \dashv x.\textbf{T}_2 \qquad \Gamma, x : \textbf{val} \mid \textbf{E}; \textbf{L} \mid \textbf{K}; \textbf{T}_2, \textbf{T} \vdash F}{\Gamma \mid \textbf{E}; \textbf{L} \mid \textbf{K}; \textbf{T}_1, \textbf{T} \vdash \texttt{let}\ \ x = I\ \ \texttt{in}\ \ F}$$

Type checking example

$\Gamma := x : \textbf{val}, ret : \textbf{val}, \alpha : \textbf{lft}, F : \textbf{lft}$

$\textbf{E} := F \sqsubseteq_e \alpha$

$\textbf{L} := F \sqsubseteq_l []$

$\textbf{K} := ret \triangleleft \textbf{cont}\,(F \sqsubseteq_l [];\, r.r \triangleleft \textbf{own}\,(() + \&^{\alpha}_{\textbf{mut}}\textbf{int}))$

$\textbf{T} := x \triangleleft \textbf{own}\,\&^{\alpha}_{\textbf{mut}}(() + \textbf{int})$

$$\text{S-NEW}\ \frac{}{\textbf{E}; \textbf{L} \mid \emptyset \vdash \texttt{new}(n) \dashv x.x \triangleleft \textbf{own}_n \lightning\, n}$$

$$\text{F-LET}\ \frac{\Gamma \mid \textbf{E}; \textbf{L} \mid \textbf{T}_1 \vdash I \dashv x.\textbf{T}_2 \qquad \Gamma, x : \textbf{val} \mid \textbf{E}; \textbf{L} \mid \textbf{K}; \textbf{T}_2, \textbf{T} \vdash F}{\Gamma \mid \textbf{E}; \textbf{L} \mid \textbf{K}; \textbf{T}_1, \textbf{T} \vdash \texttt{let}\ \ x = I\ \ \texttt{in}\ \ F}$$

$$\frac{\Gamma \mid \textbf{E}; \textbf{L} \mid \emptyset \vdash \texttt{new}(2) \dashv r.r \triangleleft \textbf{own} \lightning\, 2 \qquad \Gamma, r : \textbf{val} \mid \textbf{E}; \textbf{L} \mid \textbf{K}; r \triangleleft \textbf{own} \lightning\, 2, \textbf{T} \vdash F}{\Gamma \mid \textbf{E}; \textbf{L} \mid \textbf{K}; \textbf{T} \vdash \texttt{let}\ \ r = new(2)\ \ \texttt{in}\ \ F}$$

Type checking example

```
funrec option_as_mut(x) ret ret :=
```
$\{\mathbf{K} : ret \triangleleft \mathbf{cont}\,(F \sqsubseteq_I [\,]; r.r \triangleleft \mathbf{own}((\,) + \&^{\alpha}_{\mathbf{mut}}\mathbf{int})); \mathbf{T} : x \triangleleft \mathbf{own}\&^{\alpha}_{\mathbf{mut}}((\,) + \mathbf{int})\}$
```
  let r = new(2) in
```
$\{\mathbf{T} : x \triangleleft \mathbf{own}\,\&^{\alpha}_{\mathbf{mut}}((\,) + \mathbf{int}), r \triangleleft \mathbf{own} \lightning_2\}$
```
  letcont k() := delete(1,x); jump ret(r) in

  let y = *x in

  case *y of
    r :=inj 0 (); jump k()

    r :=inj 1 y.1;

    jump k()
```

Type checking example

```
funrec option_as_mut(x) ret ret :=
```
$\{\mathbf{K} : ret \lhd \mathbf{cont}\,(F \sqsubseteq_l [\,]; r.r \lhd \mathbf{own}((\mathbf{()} + \&^{\alpha}_{\mathbf{mut}}\mathbf{int})); \mathbf{T} : x \lhd \mathbf{own}\&^{\alpha}_{\mathbf{mut}}((\mathbf{()} + \mathbf{int})\}$
```
  let r = new(2) in
```
$\{\mathbf{T} : x \lhd \mathbf{own}\,\&^{\alpha}_{\mathbf{mut}}((\mathbf{()} + \mathbf{int}), r \lhd \mathbf{own}\,\lightning_2\}$
```
  letcont k() := delete(1,x); jump ret(r) in
```
$\{\mathbf{K} : ret \lhd \ldots, k \lhd \mathbf{cont}\,(F \sqsubseteq_l [\,]; r \lhd \mathbf{own}\,((\mathbf{()} + \&^{\alpha}_{\mathbf{mut}}\mathbf{int}), x \lhd \mathbf{own}\,\lightning_1)\}$
```
  let y = *x in

  case *y of
    r :==inj 0== (); jump k()

    r :==inj 1== y.1;

    jump k()
```

Type checking example

$$\Gamma := x : \textbf{val}, \mathit{ret} : \textbf{val}, r : \textbf{val}, k : \textbf{val}, \alpha : \textbf{lft}, F : \textbf{lft}$$

$$\textbf{E} := F \sqsubseteq_e \alpha$$

$$\textbf{L} := F \sqsubseteq_l []$$

$$\textbf{K} := \mathit{ret} \triangleleft \ldots, k \triangleleft \ldots$$

$$\textbf{T} := x \triangleleft \textbf{own}\ \underbrace{\&^{\alpha}_{\textbf{mut}}(() + \textbf{int})}_{\text{op}}, r \triangleleft \textbf{own}\,\lightning\,2$$

$$\text{S-DEREF}\ \frac{\textbf{E}; \textbf{L} \vdash \tau_1 \multimap^{\tau} \tau_1' \qquad \text{size}(\tau) = 1}{\textbf{E}; \textbf{L} \mid p \triangleleft \tau_1 \vdash *p \dashv x.\, p \triangleleft \tau_1', x \triangleleft \tau}$$

$$\text{F-LET}\ \frac{\Gamma \mid \textbf{E}; \textbf{L} \mid \textbf{T}_1 \vdash I \dashv x.\textbf{T}_2 \qquad \Gamma, x : \textbf{val} \mid \textbf{E}; \textbf{L} \mid \textbf{K}; \textbf{T}_2, \textbf{T} \vdash F}{\Gamma \mid \textbf{E}; \textbf{L} \mid \textbf{K}; \textbf{T}_1, \textbf{T} \vdash \texttt{let}\ x = I\ \texttt{in}\ F}$$

Type checking example

$\Gamma := x : \textbf{val}, ret : \textbf{val}, r : \textbf{val}, k : \textbf{val},$
$\quad \alpha : \textbf{lft}, F : \textbf{lft}$

$\mathbf{E} := F \sqsubseteq_e \alpha$

$\mathbf{L} := F \sqsubseteq_l []$

$\mathbf{K} := ret \lhd \ldots, k \lhd \ldots$

$\mathbf{T} := x \lhd \textbf{own}\ \underbrace{\&^{\alpha}_{\textbf{mut}}(() + \textbf{int})}_{\text{op}}, r \lhd \textbf{own}\not{\downarrow}_2$

$$\text{TREAD-OWN-MOVE}\ \frac{\text{size}(\tau) = n}{\mathbf{E}; \mathbf{L} \vdash \textbf{own}_m \tau \multimap^{\tau} \textbf{own}_m \not{\downarrow}_n}$$

$$\text{S-DEREF}\ \frac{\mathbf{E}; \mathbf{L} \vdash \tau_1 \multimap^{\tau} \tau_1' \qquad \text{size}(\tau) = 1}{\mathbf{E}; \mathbf{L} \mid p \lhd \tau_1 \vdash *p \dashv x.p \lhd \tau_1', x \lhd \tau}$$

$$\text{F-LET}\ \frac{\Gamma \mid \mathbf{E}; \mathbf{L} \mid \mathbf{T}_1 \vdash I \dashv x.\mathbf{T}_2 \qquad \Gamma, x : \textbf{val} \mid \mathbf{E}; \mathbf{L} \mid \mathbf{K}; \mathbf{T}_2, \mathbf{T} \vdash F}{\Gamma \mid \mathbf{E}; \mathbf{L} \mid \mathbf{K}; \mathbf{T}_1, \mathbf{T} \vdash \texttt{let}\ x = I\ \texttt{in}\ F}$$

Type checking example

$$\frac{\mathrm{size}(\tau) = n}{\mathbf{E};\mathbf{L} \vdash \mathbf{own}_m \tau \multimap^{\tau} \mathbf{own}_m \lightning_n} \text{ TREAD-OWN-MOVE}$$

$$\frac{\mathbf{E};\mathbf{L} \vdash \tau_1 \multimap^{\tau} \tau_1' \qquad \mathrm{size}(\tau) = 1}{\mathbf{E};\mathbf{L} \mid p \triangleleft \tau_1 \vdash *p \dashv x.p \triangleleft \tau_1', x \triangleleft \tau} \text{ S-DEREF}$$

$$\frac{\Gamma \mid \mathbf{E};\mathbf{L} \mid \mathbf{T}_1 \vdash I \dashv x.\mathbf{T}_2 \qquad \Gamma, x : \mathbf{val} \mid \mathbf{E};\mathbf{L} \mid \mathbf{K};\mathbf{T}_2,\mathbf{T} \vdash F}{\Gamma \mid \mathbf{E};\mathbf{L} \mid \mathbf{K};\mathbf{T}_1,\mathbf{T} \vdash \texttt{let } x = I \texttt{ in } F} \text{ F-LET}$$

$$\Gamma := x : \mathbf{val}, ret : \mathbf{val}, r : \mathbf{val}, k : \mathbf{val}, \alpha : \mathbf{lft}, F : \mathbf{lft}$$

$$\mathbf{E} := F \sqsubseteq_e \alpha$$

$$\mathbf{L} := F \sqsubseteq_l []$$

$$\mathbf{K} := ret \triangleleft \ldots, k \triangleleft \ldots$$

$$\mathbf{T} := x \triangleleft \mathbf{own}\ \underbrace{\&^{\alpha}_{\mathbf{mut}}(() + \mathbf{int})}_{\text{op}}, r \triangleleft \mathbf{own} \lightning_2$$

$$\frac{\dfrac{\dfrac{\mathrm{size}(\text{op}) = 1}{\mathbf{E};\mathbf{L} \vdash \mathbf{own}\,\text{op} \multimap^{\text{op}} \mathbf{own} \lightning_1} \qquad \mathrm{size}(\text{op}) = 1}{\mathbf{E};\mathbf{L} \mid x \triangleleft \mathbf{own}\,\text{op} \vdash *x \dashv y.x \triangleleft \mathbf{own} \lightning_1, y \triangleleft \text{op}} \qquad \Gamma, y : \mathbf{val} \mid \mathbf{E};\mathbf{L} \mid \mathbf{K}; x \triangleleft \mathbf{own} \lightning_1, y \triangleleft \text{op}, \mathbf{T'} \vdash F}{\Gamma \mid \mathbf{E};\mathbf{L} \mid \mathbf{K}; x \triangleleft \mathbf{own}\,\text{op}, \mathbf{T'} \vdash \texttt{let } y = *x \texttt{ in } F}$$

Type checking example

```
funrec option_as_mut(x) ret ret :=
```
$\{\mathbf{K} : ret \lhd \mathbf{cont}\,(F \sqsubseteq_l [\,]; r.\,r \lhd \mathbf{own}(() + \&^{\alpha}_{\mathbf{mut}}\mathbf{int})); \mathbf{T} : x \lhd \mathbf{own}\&^{\alpha}_{\mathbf{mut}}(() + \mathbf{int})\}$
```
  let r = new(2) in
```
$\{\mathbf{T} : x \lhd \mathbf{own}\,\&^{\alpha}_{\mathbf{mut}}(() + \mathbf{int}), r \lhd \mathbf{own}\,\lightning_2\}$
```
  letcont k() := delete(1,x); jump ret(r) in
```
$\{\mathbf{K} : ret \lhd \ldots, k \lhd \mathbf{cont}\,(F \sqsubseteq_l [\,]; r \lhd \mathbf{own}\,(() + \&^{\alpha}_{\mathbf{mut}}\mathbf{int}), x \lhd \mathbf{own}\,\lightning_1)\}$
```
  let y = *x in
```
$\{\mathbf{T} : x \lhd \mathbf{own}\,\lightning_1, r \lhd \mathbf{own}\,\lightning_2, y \lhd \&^{\alpha}_{\mathbf{mut}}(() + \mathbf{int})\}$
```
  case *y of
    r :==inj 0== (); jump k()

    r :==inj 1== y.1;

    jump k()
```

Type checking example

```
funrec option_as_mut(x) ret ret :=
```
$\{\mathbf{K} : \mathit{ret} \lhd \mathbf{cont}\,(F \sqsubseteq_l [\,];\, r.\, r \lhd \mathbf{own}(() + \&^{\alpha}_{\mathbf{mut}}\mathbf{int}));\ \mathbf{T} : x \lhd \mathbf{own}\,\&^{\alpha}_{\mathbf{mut}}(() + \mathbf{int})\}$
```
  let r = new(2) in
```
$\{\mathbf{T} : x \lhd \mathbf{own}\,\&^{\alpha}_{\mathbf{mut}}(() + \mathbf{int}),\, r \lhd \mathbf{own}\,\lightning_2\}$
```
  letcont k() := delete(1,x); jump ret(r) in
```
$\{\mathbf{K} : \mathit{ret} \lhd \ldots,\, k \lhd \mathbf{cont}\,(F \sqsubseteq_l [\,];\, r \lhd \mathbf{own}\,(() + \&^{\alpha}_{\mathbf{mut}}\mathbf{int}),\, x \lhd \mathbf{own}\,\lightning_1)\}$
```
  let y = *x in
```
$\{\mathbf{T} : x \lhd \mathbf{own}\,\lightning_1,\, r \lhd \mathbf{own}\,\lightning_2,\, y \lhd \&^{\alpha}_{\mathbf{mut}}(() + \mathbf{int})\}$
```
  case *y of
    r :=inj 0 (); jump k()
```
$\{\mathbf{T} : x \lhd \mathbf{own}\,\lightning_1,\, r \lhd \mathbf{own}\,\lightning_2,\, y.1 \lhd \&^{\alpha}_{\mathbf{mut}}\mathbf{int}\}$
```
    r :=inj 1 y.1;

    jump k()
```

Type checking example

```
funrec option_as_mut(x) ret ret :=
```
$\{\mathbf{K} : ret \lhd \mathbf{cont}(F \sqsubseteq_l [];\, r.\, r \lhd \mathbf{own}(() + \&^{\alpha}_{\mathbf{mut}}\mathbf{int}));\ \mathbf{T} : x \lhd \mathbf{own}\,\&^{\alpha}_{\mathbf{mut}}(() + \mathbf{int})\}$
```
  let r = new(2) in
```
$\{\mathbf{T} : x \lhd \mathbf{own}\,\&^{\alpha}_{\mathbf{mut}}(() + \mathbf{int}),\, r \lhd \mathbf{own}\, \lightning_2\}$
```
  letcont k() := delete(1,x); jump ret(r) in
```
$\{\mathbf{K} : ret \lhd \ldots,\, k \lhd \mathbf{cont}(F \sqsubseteq_l [];\, r \lhd \mathbf{own}(() + \&^{\alpha}_{\mathbf{mut}}\mathbf{int}),\, x \lhd \mathbf{own}\, \lightning_1)\}$
```
  let y = *x in
```
$\{\mathbf{T} : x \lhd \mathbf{own}\, \lightning_1,\, r \lhd \mathbf{own}\, \lightning_2,\, y \lhd \&^{\alpha}_{\mathbf{mut}}(() + \mathbf{int})\}$
```
  case *y of
```
```
    r :==inj 0 (); jump k()
```
$\{\mathbf{T} : x \lhd \mathbf{own}\, \lightning_1,\, r \lhd \mathbf{own}\, \lightning_2,\, y.1 \lhd \&^{\alpha}_{\mathbf{mut}}\mathbf{int}\}$
```
    r :==inj 1 y.1;
```
$\{\mathbf{T} : x \lhd \mathbf{own}\, \lightning_1,\, r \lhd \mathbf{own}(() + \&^{\alpha}_{\mathbf{mut}}\mathbf{int})\}$
```
    jump k()
```

Type checking example F-LETCONT

$\Gamma := x, ret, r : \textbf{val}, \alpha, F : \textbf{lft} \qquad \textbf{E} := F \sqsubseteq_e \alpha \qquad \textbf{L} := F \sqsubseteq_l [\,] \qquad \textbf{K} := ret \lhd \ldots$

$\textbf{T} := x \lhd \textbf{own}\ \underbrace{\&^{\alpha}_{\textbf{mut}}(() + \textbf{int})}_{\text{op}}, r \lhd \textbf{own}\,\lightning\,2$

$$\text{F-LETCONT}\ \frac{\begin{array}{c}\Gamma, k, \overline{x} : \textbf{val} \mid \textbf{E}; \textbf{L}_1 \mid \textbf{K}, k \lhd \textbf{cont}\,(\textbf{L}_1; \overline{x}.\textbf{T'}); \textbf{T'} \vdash F_1 \\ \Gamma, k : \textbf{val} \mid \textbf{E}; \textbf{L}_2 \mid \textbf{K}, k \lhd \textbf{cont}\,(\textbf{L}_1; \overline{x}.\textbf{T'}); \textbf{T} \vdash F_2\end{array}}{\Gamma \mid \textbf{E}; \textbf{L}_2 \mid \textbf{K}; \textbf{T} \vdash \texttt{letcont}\ \ k(\overline{x}) := F_1\ \ \texttt{in}\ \ F_2}$$

$$\frac{\begin{array}{c}\Gamma, k : \textbf{val} \mid \textbf{E}; \textbf{L}_1 \mid \textbf{K}, k \lhd \textbf{cont}\,(\textbf{L}_1; \textbf{T'}); \textbf{T'} \vdash F_1 \\ \Gamma, k : \textbf{val} \mid \textbf{E}; \textbf{L} \mid \textbf{K}; k \lhd \textbf{cont}\,(\textbf{L}_1; \textbf{T'}); \textbf{T} \vdash F\end{array}}{\Gamma \mid \textbf{E}; \textbf{L} \mid \textbf{K}; \textbf{T} \vdash \texttt{letcont}\ \ k() := F_1\ \ \texttt{in}\ \ F}$$

Where $F_1 :=$ `delete(1,x); jump ret(r)`,
$T' := r \lhd \textbf{own}\,(() + \&^{\alpha}_{\textbf{mut}}\textbf{int}), x \lhd \textbf{own}\,\lightning\,1$

Type checking example F-CASE-BOR

$\Gamma := x, ret, r, k, y : \mathbf{val}, \alpha, \digamma : \mathbf{lft} \quad \mathbf{E} := \digamma \sqsubseteq_e \alpha \quad \mathbf{L} := \digamma \sqsubseteq_l [\,] \quad \mathbf{K} := ret \lhd \ldots, k \lhd \ldots$

$\mathbf{T} := x \lhd \mathbf{own}\, \lightning_1, r \lhd \mathbf{own}\, \lightning_2, y \lhd \underbrace{\&^{\alpha}_{\mathbf{mut}}(() + \mathbf{int})}_{\mathrm{op}}$

$$\text{LALIVE-INCL}\ \frac{\mathbf{E}; \mathbf{L} \vdash \kappa \text{ alive} \qquad \mathbf{E}; \mathbf{L} \vdash \kappa \sqsubseteq \kappa'}{\mathbf{E}; \mathbf{L} \vdash \kappa' \text{ alive}} \qquad \text{LALIVE-LOCAL}\ \frac{\kappa \sqsubseteq_l \overline{\kappa} \in \mathbf{L} \qquad \forall i. \mathbf{E}; \mathbf{L} \vdash \overline{\kappa}_i \text{ alive}}{\mathbf{E}; \mathbf{L} \vdash \kappa \text{ alive}}$$

$$\text{F-CASE-BOR}\ \frac{\begin{array}{c}\mathbf{E}; \mathbf{L} \vdash \kappa \text{ alive} \\ \forall i. (\mathbf{E}; \mathbf{L} \mid \mathbf{K}; \mathbf{T}, p.1 \lhd \&^{\kappa}_{\mu} \tau_i \vdash F_i) \vee (\mathbf{E}; \mathbf{L} \mid \mathbf{K}; \mathbf{T}, p \lhd \&^{\kappa}_{\mu} \Sigma \overline{\tau} \vdash F_i)\end{array}}{\mathbf{E}; \mathbf{L} \mid \mathbf{K}; \mathbf{T}, p \lhd \&^{\kappa}_{\mu} \Sigma \overline{\tau} \vdash \mathtt{case}\ *p\ \mathtt{of}\ \overline{F}}$$

$$\frac{\dfrac{\dfrac{\digamma \sqsubseteq_l [\,]}{\mathbf{E}; \mathbf{L} \vdash \digamma \text{ alive}} \qquad \mathbf{E}; \mathbf{L} \vdash \digamma \sqsubseteq \alpha}{\mathbf{E}; \mathbf{L} \vdash \alpha \text{ alive}} \qquad \mathbf{E}; \mathbf{L} \mid \mathbf{K}; \mathbf{T}, y.1 \lhd \&^{\alpha}_{\mathbf{mut}}() \vdash F_0 \qquad \mathbf{E}; \mathbf{L} \mid \mathbf{K}; \mathbf{T}, y.1 \lhd \&^{\alpha}_{\mathbf{mut}}\mathbf{int} \vdash F_1}{\mathbf{E}; \mathbf{L} \mid \mathbf{K}; \mathbf{T}, y \lhd \mathrm{op} \vdash \mathtt{case}\ *y\ \mathtt{of}\ \overline{F}}$$

Type checking example case **int**

$\Gamma := x, ret, r, k, y : \textbf{val}, \alpha, F : \textbf{lft} \quad \textbf{E} := F \sqsubseteq_e \alpha \quad \textbf{L} := F \sqsubseteq_l [\,] \quad \textbf{K} := ret \triangleleft \ldots, k \triangleleft \ldots$

$\textbf{T} := x \triangleleft \textbf{own} ↯_1, r \triangleleft \textbf{own} ↯_2, y.1 \triangleleft \&^{\alpha}_{\textbf{mut}}\textbf{int}$

$$\text{TWRITE-OWN}\ \frac{\text{size}(\tau) = \text{size}(\tau')}{\textbf{E}; \textbf{L} \vdash \textbf{own}_n \tau' \multimap^{\tau} \textbf{own}_n \tau}$$

$$\text{S-SUM-ASSGN}\ \frac{\overline{\tau}_i = \tau \qquad \tau_1 \multimap^{\Sigma\overline{\tau}} \tau_1'}{\textbf{E}; \textbf{L} \mid p_1 \triangleleft \tau_1, p_2 \triangleleft \tau \vdash p_1 \overset{\text{inj } i}{:==} p_2 \dashv p_1 \triangleleft \tau_1'}$$

$$\frac{\&^{\alpha}_{\textbf{mut}}\textbf{int} = \&^{\alpha}_{\textbf{mut}}\textbf{int} \qquad \dfrac{\text{size}(\textbf{()} + \&^{\alpha}_{\textbf{mut}}\textbf{int}) = \text{size}(↯_2)}{\textbf{own}\, ↯_2 \multimap^{\textbf{()}+\&^{\alpha}_{\textbf{mut}}\textbf{int}} \textbf{own}\,(\textbf{()} + \&^{\alpha}_{\textbf{mut}}\textbf{int})}}{\textbf{E}; \textbf{L} \mid r \triangleleft \textbf{own}\, ↯_2, y.1 \triangleleft \&^{\alpha}_{\textbf{mut}}\textbf{int} \vdash r \overset{\text{inj } 1}{:==} y.1 \dashv r \triangleleft \textbf{own}\,(\textbf{()} + \&^{\alpha}_{\textbf{mut}}\textbf{int})}$$

Type checking example F-JUMP

$\Gamma := x, ret, r, k, y : \textbf{val}, \alpha, \digamma : \textbf{lft} \quad \textbf{E} := \digamma \sqsubseteq_e \alpha \quad \textbf{L} := \digamma \sqsubseteq_l [\,] \quad \textbf{K} := ret \lhd \ldots, k \lhd \ldots$

$\textbf{T} := x \lhd \textbf{own} \not\downarrow_1, r \lhd \textbf{own}\,(() + \&^{\alpha}_{\textbf{mut}}\textbf{int})$

$$\text{C-PERM}\;\frac{\textbf{T'} \text{ is a permutation of } \textbf{T}}{\textbf{E}; \textbf{L} \vdash \textbf{T} \overset{\text{ctx}}{\Longrightarrow} \textbf{T'}}$$

$$\text{F-JUMP}\;\frac{\textbf{E}; \textbf{L} \vdash \textbf{T} \Rightarrow \textbf{T'}[\overline{y}/\overline{x}]}{\textbf{E}; \textbf{L} \mid k \lhd \textbf{cont}\,(\textbf{L}; \overline{x}.\textbf{T'}); \textbf{T} \vdash \texttt{jump}\ \ k(\overline{y})}$$

$$\frac{\dfrac{\textbf{T'} \text{ is a permutation of } \textbf{T}}{\textbf{E}; \textbf{L} \vdash \textbf{T} \Rightarrow \textbf{T'}}}{\textbf{E}; \textbf{L} \mid k \lhd \textbf{cont}\,(\digamma \sqsubseteq_l [\,]; \textbf{T'}); \textbf{T} \vdash \texttt{jump}\ \ k()}$$

Where $T' := r \lhd \textbf{own}\,(() + \&^{\alpha}_{\textbf{mut}}\textbf{int}), x \lhd \textbf{own} \not\downarrow_1$

Type checking example delete

$\Gamma := x, ret, r, k : \textbf{val}, \alpha, F : \textbf{lft} \quad \textbf{E} := F \sqsubseteq_e \alpha \quad \textbf{L} := F \sqsubseteq_l [\,] \quad \textbf{K} := ret \triangleleft \ldots, k \triangleleft \ldots$

$\textbf{T} := r \triangleleft \textbf{own}\,(\textbf{()} + \&^{\alpha}_{\textbf{mut}}\textbf{int}), x \triangleleft \textbf{own}\ \lightning_1$

$$\text{S-DELETE}\ \frac{n = \text{size}(\tau)}{\textbf{E}; \textbf{L} \mid p \triangleleft \textbf{own}_n\, \tau \vdash \texttt{delete}(n, p) \dashv \emptyset}$$

$$\frac{1 = \text{size}(\lightning_1)}{\textbf{E}; \textbf{L} \mid x \triangleleft \textbf{own}\ \lightning_1 \vdash \texttt{delete}(1, x) \dashv \emptyset}$$

Type checking example jump ret(r)

$\Gamma := x, ret, r, k, y : \textbf{val}, \alpha, F : \textbf{lft} \quad \textbf{E} := F \sqsubseteq_e \alpha \quad \textbf{L} := F \sqsubseteq_l [\,] \quad \textbf{K} := ret \lhd \ldots, k \lhd \ldots$

$\textbf{T} := r \lhd \textbf{own}\,(() + \&^{\alpha}_{\textbf{mut}}\textbf{int})$

$$\text{C-PERM}\;\frac{\textbf{T'} \text{ is a permutation of } \textbf{T}}{\textbf{E}; \textbf{L} \vdash \textbf{T} \stackrel{\text{ctx}}{\Longrightarrow} \textbf{T'}}$$

$$\text{F-JUMP}\;\frac{\textbf{E}; \textbf{L} \vdash \textbf{T} \Rightarrow \textbf{T'}[\overline{y}/\overline{x}]}{\textbf{E}; \textbf{L} \mid k \lhd \textbf{cont}\,(\textbf{L}; \overline{x}.\textbf{T'}); \textbf{T} \vdash \texttt{jump}\ \ k(\overline{y})}$$

$$\frac{\dfrac{\textbf{T'} \text{ is a permutation of } \textbf{T}}{\textbf{E}; \textbf{L} \vdash \textbf{T} \Rightarrow \textbf{T}[r/r]}}{\textbf{E}; \textbf{L} \mid ret \lhd \textbf{cont}\,(F \sqsubseteq_l [\,]; r.\textbf{T}); \textbf{T} \vdash \texttt{jump}\ \ ret(r)}$$

Where $T' := r \lhd \textbf{own}\,(() + \&^{\alpha}_{\textbf{mut}}\textbf{int}), x \lhd \textbf{own}\ \lightning_1$