From *Linearity* to **Borrowing**

Andrew Wagner*, Olek Gierczak, Brianna Marshall, John Li*, Amal Ahmed*
Northeastern University

OOPSLA 2025, Singapore

*Attending—Come chat with us!

From *Linearity* to **Borrowing**

This is not a *Rust* talk

How can we isolate **borrowing** as a language feature …

From *Linearity* to **Borrowing**

This is not a *Rust* talk

How can we isolate **borrowing** as a language feature

and develop it as an extension of *linearity*?

From *Linearity* to **Borrowing**

This is not a *Rust* talk

Contributions

1. A lightweight borrowing extension for the linear lambda calculus with references.
    - No new syntax or operational semantics.
    - Linear typing works “as usual.”

Contributions

1. A lightweight borrowing extension for the linear lambda calculus with references.
   - No new syntax or operational semantics.
   - Linear typing works “as usual.”

2. An incremental development of distinct borrowing features.

   Linear $\lambda_{\texttt{Ref}}$ ➔ Imm ➔ Lexical Lifetimes ➔ Mut ➔ Reborrows

Contributions

1. A lightweight borrowing extension for the linear lambda calculus with references.
   - No new syntax or operational semantics.
   - Linear typing works “as usual.”

2. An incremental development of distinct borrowing features.

   Linear $\lambda_{\texttt{Ref}}$ ➔ Imm ➔ Lexical Lifetimes ➔ Mut ➔ Reborrows

3. A layered soundness proof ensuring memory safety, leak freedom, & termination.

Contributions

1. A lightweight borrowing extension for the linear lambda calculus with references.
   - No new syntax or operational semantics.
   - Linear typing works “as usual.”

2. An incremental development of distinct borrowing features.

   Linear $\lambda_{\texttt{Ref}}$ ➔ Imm ➔ Lexical Lifetimes ➔ Mut ➔ Reborrows

3. A layered soundness proof ensuring memory safety, leak freedom, & termination.

Level 0: Linear References

$\Gamma \vdash e : T$ Expression e has type T in a *linear* context Γ

Manually-Managed Memory

"$\texttt{Box}\langle T \rangle$"

- $\vdash \texttt{alloc} : T \multimap \texttt{Ref}\ T$
- $\vdash \texttt{free} : \texttt{Ref}\ T \multimap T$

Level 0: Linear References

$\Gamma \vdash e : T$ Expression e has type T in a *linear* context Γ

Manually-Managed Memory

"$\texttt{Box}\langle T \rangle$"

- $\bullet \vdash \texttt{alloc} : T \multimap \texttt{Ref}\ T$
- $\bullet \vdash \texttt{free} : \texttt{Ref}\ T \multimap T$

~~Weakening~~

$$\frac{}{x : T \vdash x : T} \qquad \frac{}{\bullet \vdash () : \texttt{Unit}}$$

Axioms are *precise* about contexts

Don't *forget* variables ➔ No leaks!

Level 0: Linear References

$\Gamma \vdash e : T$ Expression e has type T in a *linear* context Γ

Manually-Managed Memory

"$\texttt{Box}\langle T \rangle$"

- $\bullet \vdash \texttt{alloc} : T \multimap \texttt{Ref}\ T$
- $\bullet \vdash \texttt{free} : \texttt{Ref}\ T \multimap T$

~~Weakening~~

$$\frac{}{x : T \vdash x : T} \qquad \frac{}{\bullet \vdash () : \texttt{Unit}}$$

Axioms are *precise* about contexts

Don't *forget* variables ➔ No leaks!

~~Contraction~~

"Use of moved value"

$$\frac{\Gamma_1 \vdash e_1 : T_1 \quad \Gamma_2 \vdash e_2 : T_2}{\Gamma_1 \uplus \Gamma_2 \vdash (e_1, e_2) : T_1 \otimes T_2}$$

Contexts are *split* between subexpressions

Don't *dupl.* variables ➔ No use-after-free!

Level 1: Immutable Borrows

$\boxed{\Gamma \vdash e : T}$ Expression e has type T in a *linear* context Γ

Same judgment!

Level 1: Immutable Borrows

$\boxed{\Gamma \vdash e : T}$ Expression e has type T in a *linear* context Γ

Same judgment!

- $\vdash \texttt{dupl} : \texttt{Imm}\ T \multimap \texttt{Imm}\ T \otimes \texttt{Imm}\ T$
- $\vdash \texttt{forget} : \texttt{Imm}\ T \multimap \texttt{Unit}$

No `free` for `Imm`!

Level 1: Immutable Borrows

$\Gamma \vdash e : T$ Expression e has type T in a *linear* context Γ

Same judgment!

- $\vdash \texttt{dupl} : \texttt{Imm}\ T \multimap \texttt{Imm}\ T \otimes \texttt{Imm}\ T$
- $\vdash \texttt{forget} : \texttt{Imm}\ T \multimap \texttt{Unit}$

No `free` for `Imm`!

- $\vdash \texttt{withbor} : \texttt{Ref}\ T_1 \multimap (\texttt{Imm}\ T_1 \multimap []T_2) \multimap (\texttt{Ref}\ T_1) \otimes T_2$

Start with owned linear reference

Temporarily exchange for borrow

Result must *outlive* borrows

Take back ownership

Level 1: Outlives Modality

[]-Intro

$$\frac{\Gamma \vdash e : T \quad \Gamma \sqsupseteq \texttt{Imm}}{\Gamma \vdash e : []T}$$

Γ "outlives" all $\texttt{Imm}$ borrows

[]-Elim

$$\frac{\Gamma \vdash e : []T}{\Gamma \vdash e : T}$$

Erasable

Level 1: Outlives Modality

[]-Intro

$$\frac{\Gamma \vdash e : T \quad \Gamma \sqsupset \texttt{Imm}}{\Gamma \vdash e : []T}$$

Γ "outlives" all Imm borrows

$T \sqsupset \texttt{Imm}$

[]-Elim

$$\frac{\Gamma \vdash e : []T}{\Gamma \vdash e : T}$$

Erasable

Level 1: Outlives Modality

[]-Intro

$$\frac{\Gamma \vdash e : T \quad \Gamma \sqsupset \texttt{Imm}}{\Gamma \vdash e : []T}$$

Γ "outlives" all Imm borrows

[]-Elim

$$\frac{\Gamma \vdash e : []T}{\Gamma \vdash e : T}$$

Erasable

$T \sqsupset \texttt{Imm}$

Base types

$$\frac{}{\texttt{Unit} \sqsupset \texttt{Imm}}$$

Compound types inherit

$$\frac{T \sqsupset \texttt{Imm}}{\texttt{Ref}\ T \sqsupset \texttt{Imm}}$$

$$\frac{T_1 \sqsupset \texttt{Imm} \quad T_2 \sqsupset \texttt{Imm}}{T_1 \otimes T_2 \sqsupset \texttt{Imm}}$$

Level 1: Outlives Modality

[]-Intro

$$\frac{\Gamma \vdash e : T \quad \Gamma \sqsupset \texttt{Imm}}{\Gamma \vdash e : []T}$$

Γ "outlives" all Imm borrows

[]-Elim

$$\frac{\Gamma \vdash e : []T}{\Gamma \vdash e : T}$$

Erasable

$T \sqsupset \texttt{Imm}$

Base types

$$\frac{}{\texttt{Unit} \sqsupset \texttt{Imm}}$$

Compound types inherit

$$\frac{T \sqsupset \texttt{Imm}}{\texttt{Ref}\ T \sqsupset \texttt{Imm}}$$

$$\frac{T_1 \sqsupset \texttt{Imm} \quad T_2 \sqsupset \texttt{Imm}}{T_1 \otimes T_2 \sqsupset \texttt{Imm}}$$

$$\frac{}{[]T \sqsupset \texttt{Imm}}$$

Safe by []-Intro!

Level 1: Outlives Modality

[]-Intro

$$\frac{\Gamma \vdash e : T \quad \Gamma \sqsupset \texttt{Imm}}{\Gamma \vdash e : []T}$$

Γ "outlives" all Imm borrows

[]-Elim

$$\frac{\Gamma \vdash e : []T}{\Gamma \vdash e : T}$$

Erasable

$T \sqsupset \texttt{Imm}$

Base types

$$\frac{}{\texttt{Unit} \sqsupset \texttt{Imm}}$$

Compound types inherit

$$\frac{T \sqsupset \texttt{Imm}}{\texttt{Ref}\ T \sqsupset \texttt{Imm}}$$

$$\frac{T_1 \sqsupset \texttt{Imm} \quad T_2 \sqsupset \texttt{Imm}}{T_1 \otimes T_2 \sqsupset \texttt{Imm}}$$

$$\frac{}{[]T \sqsupset \texttt{Imm}}$$

Safe by []-Intro!

No rules for Imm or $\multimap$

Level 2: Lifetimes

$\boxed{\Delta; \Gamma \vdash e : T}$ Δ an unrestricted *lifetime ordering context*, Γ linear typing context

Lifetimes allow different borrows to be distinguished

Level 2: Lifetimes

$\Delta; \Gamma \vdash e : T$ Δ an unrestricted *lifetime ordering context*, Γ linear typing context

Lifetimes allow different borrows to be distinguished

$$\Delta; \bullet \vdash \texttt{withbor} : \texttt{Ref}\ T_1 \multimap (\forall a \sqsubset \Delta\,.\ \texttt{Imm}\ a\ T_1 \multimap [a]T_2) \multimap (\texttt{Ref}\ T_1) \otimes T_2$$

Fresh lifetime a shorter than all others

Tag borrow with a

Result must outlive borrows at a

Level 2: Lifetimes

$\Delta; \Gamma \vdash e : T$ Δ an unrestricted *lifetime ordering context*, Γ linear typing context

Lifetimes allow different borrows to be distinguished

$$\Delta; \bullet \vdash \texttt{withbor} : \texttt{Ref}\ T_1 \multimap (\forall a \sqsubset \Delta\,.\ \texttt{Imm}\ a\ T_1 \multimap [a]T_2) \multimap (\texttt{Ref}\ T_1) \otimes T_2$$

Fresh lifetime a shorter than all others

Tag borrow with a

Result must outlive borrows at a

$[a]$-Intro  "T : 'a"

$$\frac{\Delta; \Gamma \vdash e : T \quad \Delta \vdash \Gamma \sqsupset a}{\Delta; \Gamma \vdash e : [a]T}$$

Γ "outlives" all a borrows

Level 2: Lifetimes

$\Delta; \Gamma \vdash e : T$ Δ an unrestricted *lifetime ordering context*, Γ linear typing context

Lifetimes allow different borrows to be distinguished

$$\Delta; \bullet \vdash \texttt{withbor} : \texttt{Ref}\ T_1 \multimap (\forall a \sqsubset \Delta\,.\ \texttt{Imm}\ a\ T_1 \multimap [a]T_2) \multimap (\texttt{Ref}\ T_1) \otimes T_2$$

Fresh lifetime a shorter than all others

Tag borrow with a

Result must outlive borrows at a

$[a]$-Intro  "T : 'a"

$$\frac{\Delta; \Gamma \vdash e : T \quad \Delta \vdash \Gamma \sqsupset a}{\Delta; \Gamma \vdash e : [a]T}$$

Γ "outlives" all a borrows

$\Delta \vdash T \sqsupset \mathtt{a}$

$$\frac{\Delta \vdash b \sqsupset a}{\Delta \vdash \texttt{Imm}\ b\ T \sqsupset a}$$

From *Linear* Typing to **Borrow** Typing

Type System

~~*Contraction*~~, ~~*Weakening*~~

Imm → Lexical Lifetimes → Mut → Reborrows

Separation Logic

Semantic Model

From *Linear* Typing to **Borrow** Typing

Type System

~~*Contraction*~~, ~~*Weakening*~~

Imm → Lexical Lifetimes → Mut → Reborrows

Separation Logic

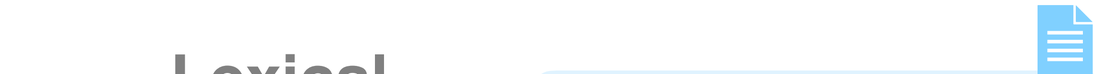
Semantic Model

From *Linear* Sep. Logic to **Borrow** Sep. Logic

Type System

~~*Contraction*~~, ~~*Weakening*~~

Imm → Lexical Lifetimes → Mut → Reborrows



Separation Logic

Semantic Model

Separation Logic

Frame

$$\frac{\{Q\}e\{P_2\}}{\{P_1 \star Q\}e\{P_1 \star P_2\}}$$

Start with P_1 owned

➔ Temporarily *ignore* P_1 during e

➔ Reclaim ownership of P_1 after e

Separation Logic

Frame

$$\frac{\{Q\}e\{P_2\}}{\{P_1 \star Q\}e\{P_1 \star P_2\}}$$

Start with P_1 owned

➔ Temporarily *ignore* P_1 during e

➔ Reclaim ownership of P_1 after e

$$\texttt{withbor} : \texttt{Ref}\ T_1 \multimap (\forall a\,.\,\texttt{Mut}\ a\ T_1 \multimap [a]T_2) \multimap \texttt{Ref}\ T_1 \otimes T_2$$

Borrowing Separation Logic

Frame

$$\frac{\{Q\}e\{P_2\}}{\{P_1 \star Q\}e\{P_1 \star P_2\}}$$

$$\texttt{withbor} : \texttt{Ref}\ T_1 \multimap (\forall a\,.\,\texttt{Mut}\ a\ T_1 \multimap [a]T_2) \multimap \texttt{Ref}\ T_1 \otimes T_2$$

Borrow Frame

$$\frac{\forall\, a\,.\,\{\ell \mapsto \mathit{Mut}\, a\, P_1 \star Q\}e\{[a]P_2\}}{\{\ell \mapsto P_1 \star Q\}e\{\ell \mapsto P_1 \star P_2\}}$$

Temporarily establish invariant P_1 for a

Borrowing Separation Logic

Frame

$$\frac{\{Q\}e\{P_2\}}{\{P_1 \star Q\}e\{P_1 \star P_2\}}$$

$$\texttt{withbor} : \texttt{Ref}\ T_1 \multimap (\forall a\,.\,\texttt{Mut}\ a\ T_1 \multimap [a]T_2) \multimap \texttt{Ref}\ T_1 \otimes T_2$$

Borrow Frame

$$\frac{\forall\, a\,.\,\{\ell \mapsto \mathit{Mut}\, a\, P_1 \star Q\}e\{[a]P_2\}}{\{\ell \mapsto P_1 \star Q\}e\{\ell \mapsto P_1 \star P_2\}}$$

Temporarily establish invariant P_1 for a

Borrow Anti-Frame  "unsafe"

$$\frac{\{\ell \mapsto P_1 \star Q\}e\{\ell \mapsto P_1 \star P_2\}}{\{\ell \mapsto \mathit{Mut}\, a\, P_1 \star Q\}e\{P_2\}}$$

Temporarily break and reestablish invariant P_1

Borrowing Separation Logic

Frame

$$\frac{\{Q\}e\{P_2\}}{\{P_1 \star Q\}e\{P_1 \star P_2\}}$$

$$\texttt{withbor} : \texttt{Ref}\ T_1 \multimap (\forall a.\, \texttt{Mut}\ a\ T_1 \multimap [a]T_2) \multimap \texttt{Ref}\ T_1 \otimes T_2$$

Borrow Frame

ESOP 2017

Temporary Read-Only Permissions for Separation Logic

Arthur Charguéraud and François Pottier

Inria*

Abstract. We present an extension of Separation Logic with a general mechanism for temporarily converting any assertion (or "permission") to a read-only form. No accounting is required: our read-only permissions can be freely duplicated and discarded. We argue that, in circumstances

Borrow Anti-Frame

"unsafe"

LICS 2008

Hiding local state in direct style: a higher-order anti-frame rule

François Pottier
INRIA

Abstract

Separation logic involves two dual forms of modularity: local reasoning makes part of the store invisible within *a static scope, whereas hiding local state makes part of the store invisible* outside *a static scope. In the recent literature, both idioms are explained in terms of a higher-order frame rule. I point out that this approach to hiding local state imposes continuation-passing style, which is impractical. Instead, I introduce a higher-order* anti-frame rule*, which permits hiding local state in direct style. I formal-*

On hidden state One often designs a piece of software so that its implementation is imperative and relies on an internal state, but its specification does not betray this fact. By this, I do not mean that the state appears under an abstract type in the specification, so that clients do not have access to its concrete representation. I mean that the very existence of an internal state is not revealed in the specification, so that clients have no knowledge whatsoever of it. A typical example is that of a memory manager [9, 2, 7]: no knowledge of the manager's internal free list should be necessary when reasoning about a client.

From *Linear* Sep. Logic to **Borrow** Sep. Logic

Type System

~~*Contraction*~~, ~~*Weakening*~~ → **Imm → Lexical Lifetimes → Mut → Reborrows**

Separation Logic

Frame Rule → **Borrow Frame & Anti-Frame Rules, Semantic Typing**

Semantic Model

From *Linear* Sep. Logic to **Borrow** Sep. Logic

Type System

~~*Contraction*~~, ~~*Weakening*~~ → Imm → Lexical Lifetimes → Mut → Reborrows

Separation Logic

Frame Rule → **Borrow Frame & Anti-Frame Rules, Semantic Typing**

Semantic Model

From *Linear* Semantics to **Borrow** Semantics

Type System

~~*Contraction*~~, ~~*Weakening*~~ → Imm → Lexical Lifetimes → Mut → Reborrows

Separation Logic

Frame Rule → Borrow Frame & Anti-Frame Rules, Semantic Typing

Semantic Model

Semantic Model: Resources (ρ)

Linear Resources

Disjoint Union of Heap Fragments

Semantic Model: Resources (ρ)

Linear Resources

Disjoint Union of Heap Fragments

Owned Ref

Exclusive:
Arbitrary updates

Semantic Model: Resources (ρ)

Linear Resources

Disjoint Union of Heap Fragments

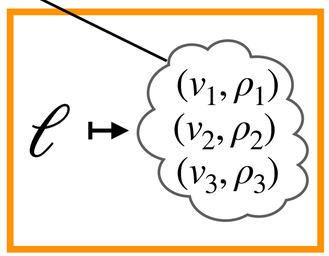
Owned Ref

$\ell \longmapsto v$

Exclusive:
Arbitrary updates

Range of legal states

Mut



Temporarily exclusive:
Type-preserving updates

Semantic Model: Resources (ρ)

Linear Resources

Disjoint Union of Heap Fragments

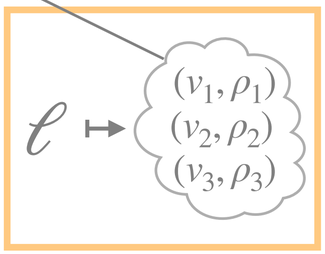
Owned Ref

$\ell \longmapsto v$

Exclusive:
Arbitrary updates

Range of legal states

Mut



Temporarily exclusive:
Type-preserving updates

In nested references, shared "dominates" exclusive 😵‍💫

Imm

$\ell \longmapsto (v, \rho)$

Temporarily shared:
No updates

Semantic Model: Locality

Frame Preservation

If $(\rho, e) \rightarrow^* (\rho', e')$, then $(\rho_F \bullet \rho, e) \rightarrow^* (\rho_F \bullet \rho', e')$ for all ρ_F

"Baking in" the *frame rule*

Semantic Model: Locality

Frame Preservation

If $(\rho, e) \to^* (\rho', e')$, then $(\rho_F \bullet \rho, e) \to^* (\rho_F \bullet \rho', e')$ for all ρ_F

"Baking in" the *frame rule*

Example 😵

$\ell_a : \texttt{Mut}\ a\ (\texttt{Option}\ (\texttt{Mut}\ b\ T)), \quad \ell_b : \texttt{Mut}\ b\ T$

$(\ell_a \mapsto \texttt{Some}(\ell_b) \bullet \rho, e) \to^* (\ell_a \mapsto \texttt{None} \bullet \rho', e')$

Need to keep track of "forgotten" nested borrows

Semantic Model: Locality

Frame Preservation

If $(\rho, e) \to^* (\rho', e')$, then $(\rho_F \bullet \rho, e) \to^* (\rho_F \bullet \rho', e')$ for all ρ_F

"Baking in" the *frame rule*

Example 😵

$\ell_a : \texttt{Mut}\ a\ (\texttt{Option}\ (\texttt{Mut}\ b\ T)), \quad \ell_b : \texttt{Mut}\ b\ T$

$(\ell_a \mapsto \texttt{Some}(\ell_b) \bullet \rho, e) \to^* (\ell_a \mapsto \texttt{None} \bullet \rho', e')$

Need to keep track of "forgotten" nested borrows

Borrow Preservation 📄

If $(\rho, e) \to^* (\rho', e')$, then

reachable borrows in $\rho \approx$ reachable borrows in ρ'

"Baking in" the **borrow frame rule**

Semantic Model: Termination

Termination

$(\rho, e) \rightarrow^* (\rho', v)$ for some ρ', v

Preserved by all program logic rules

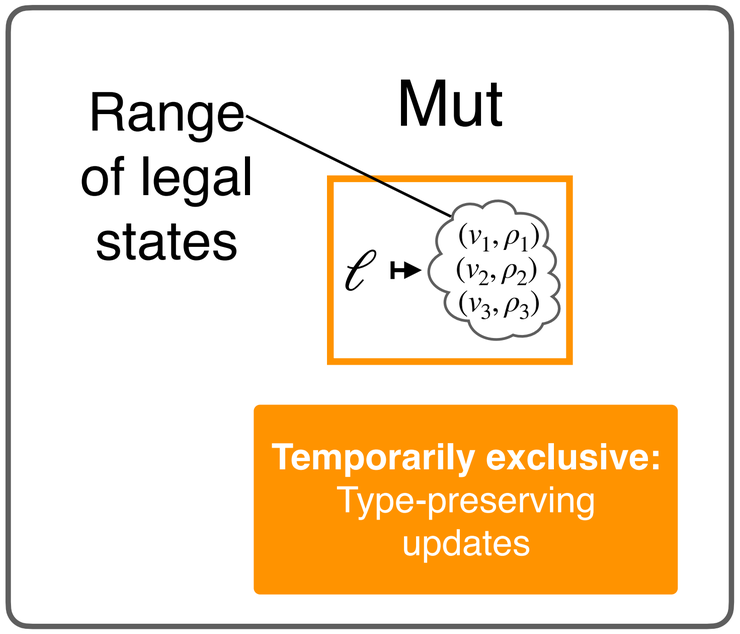
Semantic Model: Termination

Termination

$(\rho, e) \rightarrow^* (\rho', v)$ for some ρ', v

Preserved by all program logic rules

Recall ⏪



Mutable refs ➔ Circularity ➔ Step-indexing? 😰

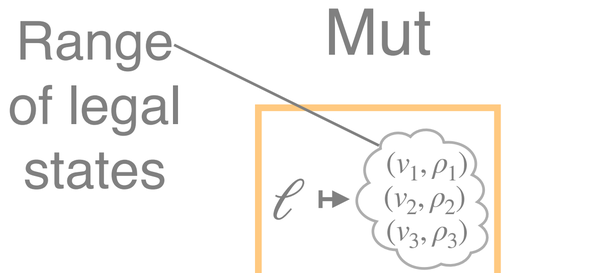
Semantic Model: Termination

Termination

$(\rho, e) \to^* (\rho', v)$ for some ρ', v

Preserved by all program logic rules

Recall ⏪



Mutable refs ➔ Circularity ➔ Step-indexing? 😰

Insight: Borrows are naturally stratified by *lifetimes*

$\texttt{Mut}\ a\ (\texttt{Mut}\ b\ T) \to a \sqsubset b$

From *Linearity* to **Borrowing**

Paper
Slides
Contact

Type System

~~*Contraction*~~,
~~*Weakening*~~

→ **Imm → Lexical Lifetimes → Mut → Reborrows**

Separation Logic

Frame Rule

→ **Borrow Frame & Anti-Frame Rules,**
Semantic Typing

Semantic Model

Termination,
Leak Freedom

→ **Lifetime Stratification,**
Borrow Preserving Updates