From Linearity to Borrowing

Andrew Wagner Olek Gierczak Brianna Marshall John Li Amal Ahmed

NEPLS, UMass Lowell, May 9, 2025

Northeastern University

Motivating Example

```
nonempty?, alpha?: Ref Str -> Bool

valid? (s: Ref Str): Bool = nonempty? s && alpha? s

main (): Bool = // purpose: validate user input
  let buf: Ref Str = readln ();
  valid? buf
```

Manual Memory Management

```
nonempty?, alpha?: Ref Str -> Bool

valid? (s: Ref Str): Bool = nonempty? s && alpha? s

main (): Bool = // purpose: validate user input
  let buf: Ref Str = readln ();
  valid? buf
  // Memcheck: 1 block definitely lost
```

Manual Memory Management

```
nonempty?, alpha?: Ref Str -> Bool

valid? (s: Ref Str): Bool = nonempty? s && alpha? s

main (): Bool = // purpose: validate user input
  let buf: Ref Str = readln ();
  free buf;
  valid? buf // Memcheck: Invalid read
```

Manual Memory Management

```
nonempty?, alpha?: Ref Str -> Bool

valid? (s: Ref Str): Bool = nonempty? s && alpha? s

main (): Bool = // purpose: validate user input
  let buf: Ref Str = readln ();
  let res = valid? buf;
  free buf
  res
```

Manual Memory Management

Linear typing prevents resource misuse by ensuring that every resource is consumed (e.g., `free`d) exactly once.

- No contraction (duplicating variables)
- No weakening (forgetting variables)

Manual Memory Management

```
nonempty?, alpha?: Ref Str -> Bool

valid? (s: Ref Str): Bool = nonempty? s && alpha? s

main (): Bool = // purpose: validate user input
  let buf: Ref Str = readln ();
  let res = valid? buf;
  free buf
  res
```

`s` and `buf` are not used linearly

Linear Typing

```
nonempty?, alpha?: Ref Str -> Ref Str * Bool

valid? (s: Ref Str): Ref Str * Bool =
  let (s', b1) = nonempty? s;
  let (s'', b2) = alpha? s';
  (s'', b1 && b2)

main (): Bool = // purpose: validate user input
  let buf: Ref Str = readln ();
  let (buf', res) = valid? buf;
  free buf'
  res
```

Capability-passing style percolates through user and library code

A borrow of a resource temporarily disables certain `access permissions` in exchange for enabling certain *structural permissions*.

A borrow of a resource temporarily disables certain `access permissions` in exchange for enabling certain *structural permissions*.

`Deallocation or Mutation`

A borrow of a resource temporarily disables certain `access permissions` in exchange for enabling certain *structural permissions*.

`Deallocation or Mutation`

Contraction or Weakening

(Duplication or Forgetting)

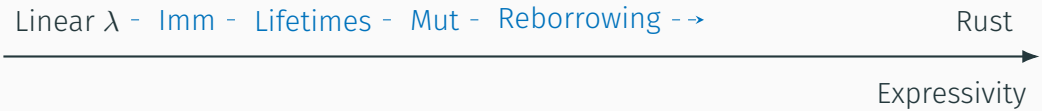
Contributions

Contributions

1. A *lightweight* borrowing extension to the linear lambda calculus.

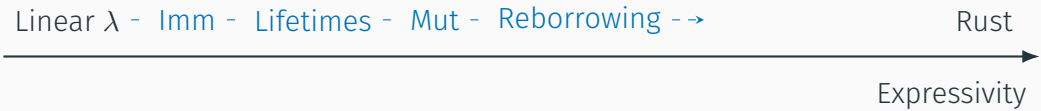
Contributions

1. A *lightweight* borrowing extension to the linear lambda calculus.
2. An *incremental* development of distinct borrowing features.

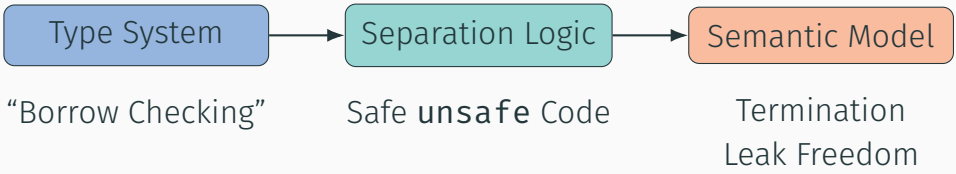


Contributions

1. A *lightweight* borrowing extension to the linear lambda calculus.
2. An *incremental* development of distinct borrowing features.



3. A new *borrowing separation logic* whose model ensures *termination*.



Immutable Borrows

$\boxed{\Gamma \vdash \mathtt{e} : \mathtt{T}}$ Γ a linear typing context

DUPL

$\vdash$ `dupl` : `Imm T -> Imm T * Imm T`

FORGET

$\vdash$ `forget` : `Imm T -> Unit`

Immutable Borrows

$\boxed{\Gamma \vdash \mathtt{e} : \mathtt{T}}$ Γ a linear typing context

DUPL

$\vdash$ `dupl` : `Imm T -> Imm T * Imm T`

FORGET

$\vdash$ `forget` : `Imm T -> Unit`

WITHBOR

$\vdash$ `withbor` : `Ref T1 -> (Imm T1 -> []T2) -> (Ref T1) * T2`

Outlives Modality ensures borrow is temporary
by requiring result to outlive all borrows

Immutable Borrows

$\boxed{\Gamma \vdash \mathtt{e} : \mathtt{T}}$ Γ a linear typing context

DUPL

$\vdash$ `dupl` : `Imm T -> Imm T * Imm T`

FORGET

$\vdash$ `forget` : `Imm T -> Unit`

WITHBOR

$\vdash$ `withbor` : `Ref T1 -> (Imm T1 -> []T2) -> (Ref T1) * T2`

OUTLIVES-INTRO

$$\frac{\Gamma \vdash \mathtt{e} : \mathtt{T} \qquad \text{“No Imms in } \Gamma\text{”}}{\Gamma \vdash \mathtt{e} : []\mathtt{T}}$$

OUTLIVES-ELIM

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

Immutable Borrows

Immutable Borrows

$$\frac{}{\texttt{Unit}\ \text{unborrowed}} \qquad \frac{\texttt{T}\ \text{unborrowed}}{\texttt{Ref T}\ \text{unborrowed}}$$

$$\frac{\texttt{T1}\ \text{unborrowed} \qquad \texttt{T2}\ \text{unborrowed}}{\texttt{T1 * T2}\ \text{unborrowed}}$$

Immutable Borrows

$$\frac{}{\texttt{Unit}\ \text{unborrowed}} \qquad \frac{\texttt{T}\ \text{unborrowed}}{\texttt{Ref T}\ \text{unborrowed}}$$

$$\frac{\texttt{T1}\ \text{unborrowed} \qquad \texttt{T2}\ \text{unborrowed}}{\texttt{T1 * T2}\ \text{unborrowed}} \qquad \frac{}{\texttt{[]T}\ \text{unborrowed}}$$

Immutable Borrows

$$\frac{}{\texttt{Unit}\ \text{unborrowed}} \qquad \frac{\texttt{T}\ \text{unborrowed}}{\texttt{Ref T}\ \text{unborrowed}}$$

$$\frac{\texttt{T1}\ \text{unborrowed} \qquad \texttt{T2}\ \text{unborrowed}}{\texttt{T1 * T2}\ \text{unborrowed}} \qquad \frac{}{\texttt{[]T}\ \text{unborrowed}}$$

No rules for `->` or `Imm`

Revising the Example

```
nonempty?, alpha?: Imm Str -> Bool

valid? (s: Imm Str): Bool =
  let (s1, s2) = dupl s;
  nonempty? s1 && alpha? s2

main (): Bool = // purpose: validate user input
  let (buf, res) = withbor (readln ()) (fun ibuf =>
    valid? ibuf);
  free buf;
  res
```

Extending the Example

```
main (): Bool = // purpose: confirm and validate user input
  let (buf1, res1) = withbor (readln ()) (fun ibuf1 =>
    let (buf2, res2) = withbor (readln ()) (fun ibuf2 =>
      if match? ibuf1 ibuf2 then Some ibuf1 else None);
    free buf2;
    some_and valid? res2);
  free buf1;
  res1
```

Confirm two inputs match and then validate

Extending the Example

```
main (): Bool = // purpose: confirm and validate user input
  let (buf1, res1) = withbor (readln ()) (fun ibuf1 =>
    let (buf2, res2) = withbor (readln ()) (fun ibuf2 =>
      if match? ibuf1 ibuf2 then Some ibuf1 else None);
    free buf2;
    some_and valid? res2);
  free buf1;
  res1
```

`res2` does not outlive *all* borrows, but it *does* outlive the borrow `buf2'`...

Lifetimes

Lifetimes allow different borrows to be distinguished

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

$$\Delta; \bullet \vdash \texttt{withbor} : \begin{cases} \texttt{Ref T1} & \texttt{->} \\ \text{Tag bor. with lifetime } \texttt{'a} \text{ shorter than all in } \Delta & \\ \forall \texttt{'a} \sqsubset \Delta.\texttt{(Imm 'a T1 -> ['a]T2)} & \texttt{->} \\ \\ \texttt{(Ref T1) * T2} & \end{cases}$$

Lifetimes

Lifetimes allow different borrows to be distinguished

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

$$\Delta; \bullet \vdash \texttt{withbor} : \begin{cases} \texttt{Ref T1} & \texttt{->} \\ \text{Tag bor. with lifetime } \texttt{'a} \text{ shorter than all in } \Delta & \\ \forall \texttt{'a} \sqsubset \Delta.\texttt{(Imm 'a T1 -> ['a]T2)} & \texttt{->} \\ \text{Only block borrows at } \texttt{'a} \text{ or shorter} & \\ \texttt{(Ref T1) * T2} & \end{cases}$$

$$\textsc{Outlives-Intro}\ \frac{\Delta; \Gamma \vdash \texttt{e} : \texttt{T} \qquad \text{“All } \texttt{Imm}\text{s in } \Gamma \text{ outlive } \texttt{'a}\text{”}}{\Delta; \Gamma \vdash \texttt{e} : \texttt{['a]T}}$$

Lifetimes

(⊐ pronounced “outlives”)

$$\frac{}{\Delta \vdash \texttt{Unit} \sqsupset \texttt{'a}} \qquad \frac{\Delta \vdash \texttt{T} \sqsupset \texttt{'a}}{\Delta \vdash \texttt{Ref T} \sqsupset \texttt{'a}} \qquad \frac{\Delta \vdash \texttt{T1} \sqsupset \texttt{'a} \quad \Delta \vdash \texttt{T2} \sqsupset \texttt{'a}}{\Delta \vdash \texttt{T1 * T2} \sqsupset \texttt{'a}}$$

Lifetimes

(⊐ pronounced “outlives”)

$$\frac{}{\Delta \vdash \texttt{Unit} \sqsupset \texttt{'a}} \qquad \frac{\Delta \vdash \texttt{T} \sqsupset \texttt{'a}}{\Delta \vdash \texttt{Ref T} \sqsupset \texttt{'a}} \qquad \frac{\Delta \vdash \texttt{T1} \sqsupset \texttt{'a} \qquad \Delta \vdash \texttt{T2} \sqsupset \texttt{'a}}{\Delta \vdash \texttt{T1 * T2} \sqsupset \texttt{'a}}$$

$$\frac{\Delta \vdash \texttt{'b} \sqsupset \texttt{'a}}{\Delta \vdash \texttt{['b]T} \sqsupset \texttt{'a}}$$

Lifetimes

(⊐ pronounced “outlives”)

$$\frac{}{\Delta \vdash \texttt{Unit} \sqsupset \texttt{'a}} \qquad \frac{\Delta \vdash \texttt{T} \sqsupset \texttt{'a}}{\Delta \vdash \texttt{Ref T} \sqsupset \texttt{'a}} \qquad \frac{\Delta \vdash \texttt{T1} \sqsupset \texttt{'a} \quad \Delta \vdash \texttt{T2} \sqsupset \texttt{'a}}{\Delta \vdash \texttt{T1 * T2} \sqsupset \texttt{'a}}$$

$$\frac{\Delta \vdash \texttt{'b} \sqsupset \texttt{'a}}{\Delta \vdash \texttt{['b]T} \sqsupset \texttt{'a}} \qquad \frac{\Delta \vdash \texttt{'b} \sqsupset \texttt{'a}}{\Delta \vdash \texttt{Imm 'b T} \sqsupset \texttt{'a}}$$

Lifetimes

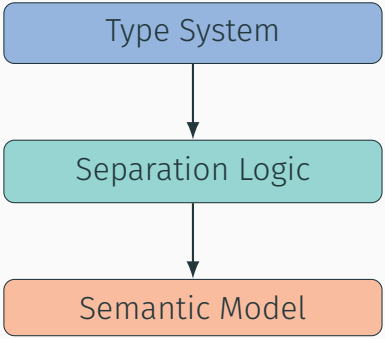
(⊐ pronounced “outlives”)

$$\frac{}{\Delta \vdash \texttt{Unit} \sqsupset \texttt{'a}} \qquad \frac{\Delta \vdash \texttt{T} \sqsupset \texttt{'a}}{\Delta \vdash \texttt{Ref T} \sqsupset \texttt{'a}} \qquad \frac{\Delta \vdash \texttt{T1} \sqsupset \texttt{'a} \qquad \Delta \vdash \texttt{T2} \sqsupset \texttt{'a}}{\Delta \vdash \texttt{T1 * T2} \sqsupset \texttt{'a}}$$

$$\frac{\Delta \vdash \texttt{'b} \sqsupset \texttt{'a}}{\Delta \vdash \texttt{['b]T} \sqsupset \texttt{'a}} \qquad \frac{\Delta \vdash \texttt{'b} \sqsupset \texttt{'a}}{\Delta \vdash \texttt{Imm 'b T} \sqsupset \texttt{'a}}$$

No rule for ->

Outline



$\Delta; \Gamma \vdash \mathsf{e} : \mathsf{T}$

?

Borrowing Separation Logic

withmut: Ref T_1 -> ∀'a.(Mut 'a T_1 -> ['a]T_2) -> Ref T_1 * T_2

Borrowing Separation Logic

```
withmut: Ref T₁ -> ∀'a.(Mut 'a T₁ -> ['a]T₂) -> Ref T₁ * T₂
```

WITHMUT

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

Borrowing Separation Logic

```
withmut: Ref T₁ -> ∀'a.(Mut 'a T₁ -> ['a]T₂) -> Ref T₁ * T₂
```

WITHMUT (BOR-FRAME)[1]

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

FRAME

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

[1] Charguéraud and Pottier, "Temporary read-only permissions for separation logic", 2017.

Borrowing Separation Logic

```
withmut: Ref T1 -> ∀'a.(Mut 'a T1 -> ['a]T2) -> Ref T1 * T2
```

WITHMUT (BOR-FRAME)[1]

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

WITHSWAP

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

```
withswap: Mut a T1 -> (T1 -> T1 * T2) -> Mut a T1 * T2
```

Borrowing Separation Logic

```
withmut: Ref T1 -> ∀'a.(Mut 'a T1 -> ['a]T2) -> Ref T1 * T2
```

WITHMUT (BOR-FRAME)[1]

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

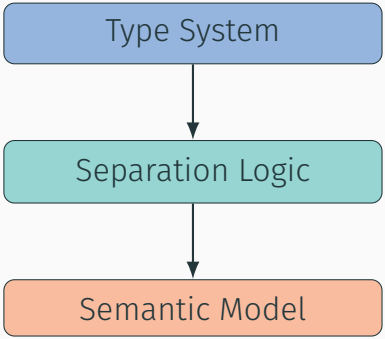
WITHSWAP (BOR-ANTI-FRAME)[2]

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

```
withswap: Mut a T1 -> (T1 -> T1 * T2) -> Mut a T1 * T2
```

[2] Pottier, “Hiding local state in direct style: a higher-order anti-frame rule”, 2008.

Outline



$$\Delta; \Gamma \vdash \mathsf{e} : \mathsf{T}$$

$$\downarrow$$

$$\{[\![\Delta]\!] \star [\![\Gamma]\!]\}\ \mathsf{e}\ \{[\![\mathsf{T}]\!]\}$$

$$\downarrow$$

$$?$$

A Model for Termination

$$\begin{aligned}\textsc{SProp} &\approx (\textsc{Loc} \rightharpoonup \textsc{Cell}) \to \mathbb{P} \\ \textsc{Cell} &\approx \mathsf{own}(\textsc{Val}) + \mathsf{imm}(\cdots) + \mathsf{mut}\big(\{\mathsf{inv} : \textsc{Val} \to \textsc{SProp}, \ldots\}\big)\end{aligned}$$

A Model for Termination

Circular!

$$\textsc{SProp} \approx (\textsc{Loc} \rightharpoonup \textsc{Cell}) \to \mathbb{P}$$
$$\textsc{Cell} \approx \mathsf{own}(\textsc{Val}) + \mathsf{imm}(\cdots) + \mathsf{mut}(\{\mathsf{inv} : \textsc{Val} \to \textsc{SProp}, \ldots\})$$

A Model for Termination

Stratifying by steps [Ahmed04] works but complicates a termination proof

$$
\begin{aligned}
\textsc{SProp}_k &\approx (\textsc{Loc} \rightharpoonup \textsc{Cell}_k) \to \mathbb{P} \\
\textsc{Cell}_k &\approx \mathsf{own}(\textsc{Val}) + \mathsf{imm}(\cdots) + \mathsf{mut}\big(\{\mathsf{step} : j < k, \mathsf{inv} : \textsc{Val} \to \textsc{SProp}_j, \ldots\}\big) \\
j, k &\in \mathbb{N}
\end{aligned}
$$

A Model for Termination

Observation: In any nested borrow `Mut a (Mut b T)`, we must have $\mathtt{a} \sqsubset \mathtt{b}$.

A Model for Termination

Stratify by lifetimes!

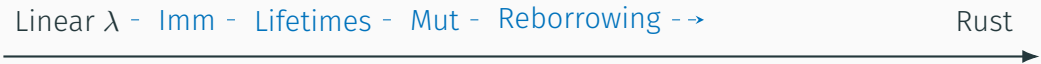
$$
\begin{aligned}
\textsc{SProp}_a &\approx (\textsc{Loc} \rightharpoonup \textsc{Cell}_a) \to \mathbb{P} \\
\textsc{Cell}_a &\approx \mathsf{own}(\textsc{Val}) + \mathsf{imm}(\cdots) + \mathsf{mut}\big(\{\mathsf{life} : b \sqsupset a, \mathsf{inv} : \textsc{Val} \to \textsc{SProp}_b, \ldots\}\big) \\
a, b &\in \textsc{Lifetime}
\end{aligned}
$$

Theorem (Adequacy)

If $\{\mathsf{emp}\}\ e\ \{v.\ v = v^\dagger\}$*, then running expression* e *in the empty heap terminates at value* $v^\dagger$ *the empty heap.*

Contributions

1. A *lightweight* borrowing extension to the linear lambda calculus.
2. An *incremental* development of distinct borrowing features.



3. A new *borrowing separation logic* whose model ensures *termination*.