Non-Step-Indexed Separation Logic with Invariants and Rust-Style Borrows

Yusuke Matsushita

Supervised by Prof. Naoki Kobayashi

January 16, 2024 — Ph.D. Thesis Defense

Program verification

Reasoning 🧠 about behaviors of

the **execution** ⚙️ of **programs** 📄

Esp. Prove absence of bugs 🐞

Program verification

Reasoning about behaviors of the **execution** of **programs**

Esp. Prove absence of bugs

Example

Type system

Commonly used & Lightweight

etc.

Program verification

Reasoning about behaviors of
the **execution** of **programs**

Esp. Prove absence of bugs

Example

Type system

Commonly used & Lightweight

TS *etc.*

Program logic

Foundational & General *(Hoare '69) etc.*

Explore sound & powerful reasoning principles

Separation logic ∗ for mutable state

Separation logic ∗ for mutable state

Mutable state **Global state that can be mutated**

Esp. Mutable objects on heap memory

Core difficulty in program reasoning *Causes real-world bugs: Use after free, …*

Separation logic ∗ for mutable state

Mutable state **Global state that can be mutated**

Esp. Mutable objects on heap memory

Core difficulty in program reasoning *Causes real-world bugs: Use after free, …*

Separation logic ∗ *(O'Hearn+ '99), (Ishitaq+ '01), …*

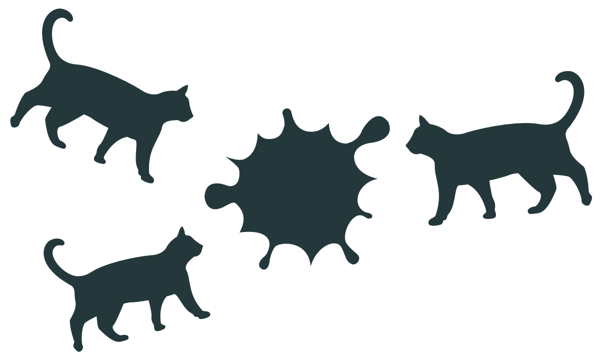
Scalable program logic for mutable state

Actively studied, de facto standard program logic for mutable state

Key idea **Use ownership to eliminate aliasing**

Propositional sharing & Problem of existing work

Propositional sharing & Problem of existing work

Big challenge **Shared mutable state in SL** * 

E.g., Mutex-guarded shared object

Propositional sharing & Problem of existing work

Big challenge **Shared mutable state in SL** ∗

E.g., Mutex-guarded shared object

Propositional sharing — *Modern SLs* *(Hobor+ '08), (Buisse+ '11),* Iris *(Jung+ '15),* *(Jung+ '18), …*

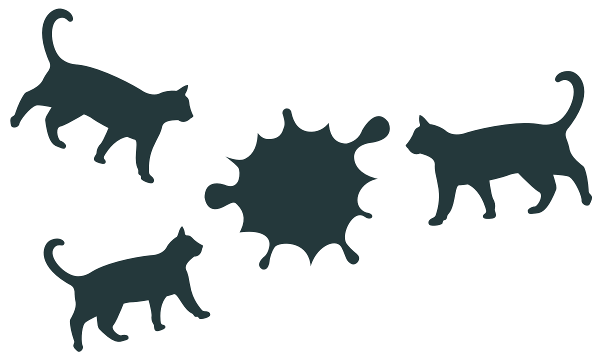
Sharing with contract by SL props

Solved challenging problems

Memory safety by Rust's ownership types (Jung+ '18),
Information-flow control (Gregersen+ '21), Purity of ST monad (Jacobs+ '22), …

Propositional sharing & Problem of existing work

Big challenge **Shared mutable state in SL** $*$



E.g., Mutex-guarded shared object

Propositional sharing

Modern SLs *(Hobor+ '08), (Buisse+ '11),*
Iris *(Jung+ '15),* *(Jung+ '18), …*

Sharing with contract by SL props

Solved challenging problems

Memory safety by Rust's ownership types (Jung+ '18),
Information-flow control (Gregersen+ '21), Purity of ST monad (Jacobs+ '22), …

Existing work **Later modality** $\triangleright$ → Can't verify **liveness**

termination etc.

High-level overview

Core contribution   Mechanization   Future applications

Core contribution of my work

Verification goals	Basic SLs	Recent SLs *Iris (Jung+ '15) etc.*	My work Nola Framework for building SLs
Liveness ♥ Termination etc.	✔	✘	✔
		Later modality ▷	**Later-free** **Syntax for SL props**
Propositional Sharing	✘	✔	✔

Separation logic ∗ *Scalable program logic for mutable state*

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic ∗

Old			Syntax for SL props	
Later modality	$\triangleright$	➜	**Later-free**	
Step-indexing	No liveness		**No step-indexing**	**✔ Liveness ♥**

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic ∗

Old

Later modality ▷

Step-indexing No liveness

➜

Syntax for SL props

Later-free

No step-indexing ✔ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic $*$

Old

Later modality $\triangleright$

Step-indexing No liveness

→

Syntax for SL props

Later-free

No step-indexing ✔ **Liveness** ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Semantic alteration

Novel general approach §4

Nola is fully mechanized as a general library

Nola is fully mechanized as a general library

Fully mechanized in Coq

Proofs are rigorously formalized & machine-checked

Nola is fully mechanized as a general library

Fully mechanized in Coq

Proofs are rigorously formalized & machine-checked

General library on Iris *(Jung+ '15)*

SL framework used in various studies
Won Alonzo Church Award '23

Can be combined with diverse Iris-based studies

Publicly available at `https://github.com/hopv/nola`

Possible future applications of my work Nola

Possible future applications of my work Nola

- **Practical verification tools w/ propositional sharing**
  - Liveness such as program termination ← Later-free
  - Support invariant & borrow in SL-based verification platforms
    Viper (Müller+ '16), *Steel* (Fromherz+ '21), …
  - Foundation for verifiers that leverage Rust's types etc.
    RustHorn (Matsushita+ '20), *Creusot* (Denis+ '22), …

Possible future applications of my work Nola

- **Practical verification tools w/ propositional sharing**
  - Liveness such as program termination ← Later-free
  - Support invariant & borrow in SL-based verification platforms
    Viper (Müller+ '16), Steel (Fromherz+ '21), …
  - Foundation for verifiers that leverage Rust's types etc.
    RustHorn (Matsushita+ '20), Creusot (Denis+ '22), …

- **Verifying program optimization algorithms**
  - Leverage Rust's types etc. ← Propositional sharing
  - Verify (fair) termination preservation ← Liveness ← Later-free
    SL such as Simuliris (Gäher+ '22), …

General background

Liveness ♥ Separation logic ∗

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic *

Old

Later modality ▷

Step-indexing No liveness

→

Syntax for SL props

Later-free

No step-indexing ✔ **Liveness** ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Semantic alteration

Novel general approach §4

Partial vs Total correctness

Two types of standard program correctness

Partial correctness **Total correctness ♥**

Partial vs Total correctness

Two types of standard program correctness

Partial correctness

e doesn't terminate with a non-Ψ result

$$\{P\}\ e\ \{\Psi\}$$

Partial Hoare triple

Total correctness ♥

e terminates with a Ψ result

$$[P]\ e\ [\Psi]$$

Total Hoare triple

Partial vs Total correctness

Two types of standard program correctness

Partial correctness	**Total correctness ♥**
e doesn't terminate with a non-Ψ result	e terminates with a Ψ result
$\{P\}\ e\ \{\Psi\}$	$[P]\ e\ [\Psi]$
Partial Hoare triple	**Total Hoare triple**

Example $\text{fun}\ \ \text{osum}(n)\ \{\ \text{if}\ \ n \neq 0\ \{\ 2\times n - 1 + \text{osum}(n-1)\ \}\ \text{else}\ \{\ 0\ \}\ \}$

$\{n \in \mathbb{Z}\}\ \text{osum}(n)\ \{\lambda v.\, v = n^2\}$ $[n \in \mathbb{N}]\ \text{osum}(n)\ [\lambda v.\, v = n^2]$

Infinite loop 🐞 $-1 \to -2 \to -3 \to \cdots$ **Terminate!** $\because$ *Induction by* $n \in \mathbb{N}$

Safety vs Liveness ♥

Two classes of program properties

Safety

Liveness ♥

Safety vs Liveness ♥

Two classes of program properties

Safety		Liveness ♥
$\{P\}\ e\ \{\Psi\}$	Examples	$[P]\ e\ [\Psi]$
Partial correctness		Total correctness
Bad things don't happen	Roughly	**Good things eventually happen**
Errors, Bad outputs, …		*Termination, …*
Coinduction	Typical proof	Induction

Safety vs Liveness ♥

Two classes of program properties

Safety		Liveness ♥
$\{P\}\ e\ \{\Psi\}$	Examples	$[P]\ e\ [\Psi]$
Partial correctness		Total correctness
Bad things don't happen	Roughly	**Good things eventually happen**
Errors, Bad outputs, …		*Termination, …*
Coinduction	Typical proof	Induction

🔥 *Significant & Challenging*

Damaged by later ▷

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic *

Old

Later modality ▷

Step-indexing No liveness

→

Syntax for SL props

Later-free

No step-indexing ✔ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Semantic alteration

Novel general approach §4

Reasoning about mutable state is hard

Example

Reasoning about mutable state is hard

Example

$$? \left[x \mapsto 0 \;\wedge\; y \mapsto 0 \right] \; x \mathrel{+}= 1;\; y \mathrel{+}= 1 \; \left[x \mapsto 1 \;\wedge\; y \mapsto 1 \right]$$

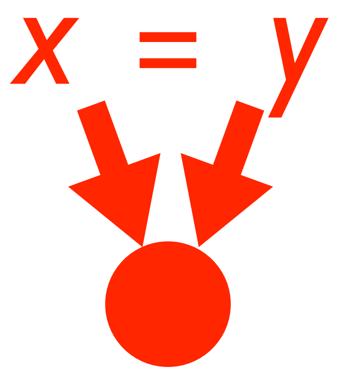
Reasoning about mutable state is hard

Example

Unsound ?

$$\cancel{\left[\, x \mapsto 0 \;\wedge\; y \mapsto 0 \,\right]\; x \mathrel{+}= 1;\; y \mathrel{+}= 1 \;\left[\, x \mapsto 1 \;\wedge\; y \mapsto 1 \,\right]}$$

Aliasing



$$\left[\, x = y \;\wedge\; x \mapsto 0 \;\wedge\; y \mapsto 0 \,\right]\; x \mathrel{+}= 1;\; y \mathrel{+}= 1 \;\left[\, x \mapsto 2 \;\wedge\; y \mapsto 2 \,\right]$$

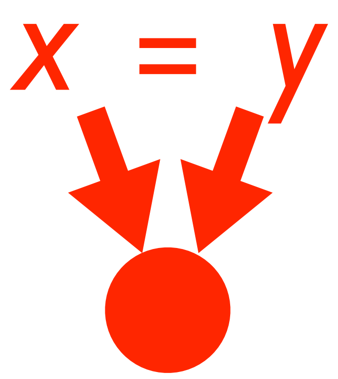
Unexpected

Reasoning about mutable state is hard

Example

Unsound $\cancel{? \left[x \mapsto 0 \ \land \ y \mapsto 0 \right] \ x \mathrel{+}= 1; \ y \mathrel{+}= 1 \ \left[x \mapsto 1 \ \land \ y \mapsto 1 \right]}$

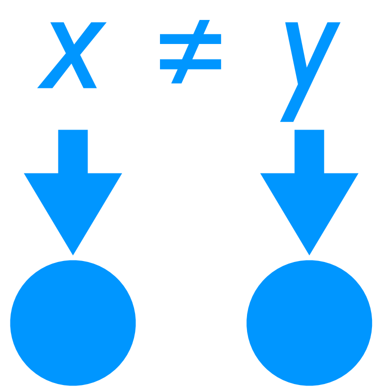
Aliasing



$$\left[x = y \ \land \ x \mapsto 0 \ \land \ y \mapsto 0 \right] \ x \mathrel{+}= 1; \ y \mathrel{+}= 1 \ \left[x \mapsto 2 \ \land \ y \mapsto 2 \right]$$

Unexpected

No aliasing



$$\left[x \neq y \ \land \ x \mapsto 0 \ \land \ y \mapsto 0 \right] \ x \mathrel{+}= 1; \ y \mathrel{+}= 1 \ \left[x \mapsto 1 \ \land \ y \mapsto 1 \right]$$

Manually eliminate aliasing → Not scalable

Basics of separation logic ∗

(O'Hearn+ '99), (Ishitaq+ '01), etc.

Separation logic ∗ **Scalable program logic for mutable state**

Basics of separation logic $\ast$

(O'Hearn+ '99), (Ishitaq+ '01), etc.

Separation logic $\ast$ **Scalable program logic for mutable state**

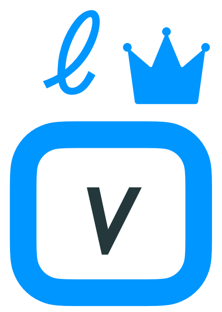
E.g., $\left[\, x \mapsto 0 \;\ast\; y \mapsto 0 \,\right]\; x \mathrel{+}= 1;\; y \mathrel{+}= 1 \;\left[\, x \mapsto 1 \;\ast\; y \mapsto 1 \,\right]$ *No aliasing by $\ast$*

Basics of separation logic $*$

(O'Hearn+ '99), (Ishitaq+ '01), etc.

Separation logic $*$ **Scalable program logic for mutable state**

E.g., $\left[x \mapsto 0 \ * \ y \mapsto 0 \right] \ x \mathrel{+}= 1;\ y \mathrel{+}= 1 \ \left[x \mapsto 1 \ * \ y \mapsto 1 \right]$ *No aliasing by $*$*

Points-to token $\ell \mapsto v$ 

Ownership of memory cell

Exclusive access right to mutable state

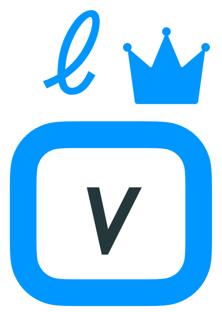
Basics of separation logic ✻

(O'Hearn+ '99), (Ishitaq+ '01), etc.

Separation logic ✻ **Scalable** program logic for mutable state

E.g., $\left[\, x \mapsto 0 \;*\; y \mapsto 0 \,\right]\; x \mathrel{+}= 1;\; y \mathrel{+}= 1 \;\left[\, x \mapsto 1 \;*\; y \mapsto 1 \,\right]$ *No aliasing by ✻*

Points-to token $\ell \mapsto v$



Ownership of memory cell

Exclusive access right to mutable state

Separating conjunction $P * Q$

Disjoint ownership

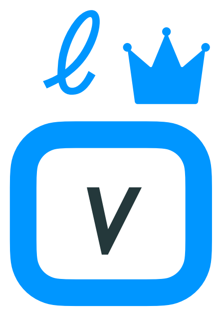
Basics of separation logic ✻

(O'Hearn+ '99), (Ishitaq+ '01), etc.

Separation logic ✻ **Scalable program logic for mutable state**

E.g., $\left[\, x \mapsto 0 \ * \ y \mapsto 0 \,\right] \; x \mathrel{+}= 1;\ y \mathrel{+}= 1 \; \left[\, x \mapsto 1 \ * \ y \mapsto 1 \,\right]$ *No aliasing by ✻*

Points-to token $\ell \mapsto v$



Ownership of memory cell

Exclusive access right to mutable state

Separating conjunction $P * Q$

Disjoint ownership

$$\ell \mapsto v \ * \ \ell' \mapsto v' \ \vDash \ \ell \neq \ell'$$

No aliasing

$$\left[\, \ell \mapsto v \ * \ P \,\right] \; \ell \leftarrow w \; \left[\, \ell \mapsto w \ * \ P \,\right]$$

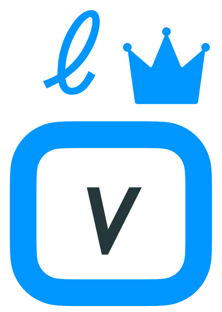
Retained

Basics of separation logic ✻

(O'Hearn+ '99), (Ishitaq+ '01), etc.

Separation logic ✻ **Scalable program logic for mutable state**

E.g., $\left[\, x \mapsto 0 \;*\; y \mapsto 0 \,\right] \; x \mathrel{+}= 1;\; y \mathrel{+}= 1 \; \left[\, x \mapsto 1 \;*\; y \mapsto 1 \,\right]$ *No aliasing by ✻*

Points-to token $\ell \mapsto v$ 

Ownership of memory cell
Exclusive access right to mutable state

Separating conjunction $P * Q$ 

Disjoint ownership

$\ell \mapsto v \;*\; \ell' \mapsto v' \;\models\; \ell \neq \ell'$

No aliasing

$\left[\, \ell \mapsto v \;*\; P \,\right] \; \ell \leftarrow w \; \left[\, \ell \mapsto w \;*\; P \,\right]$

Retained

Concurrency **Thread-local reasoning**

$$\frac{[P]\; e\; [Q] \qquad [P']\; e'\; [Q']}{[P * P']\; e \parallel e'\; [Q * Q']}$$

Separation between threads

Direct background

Propositional sharing

Later modality

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic ∗

Old

Later modality ▷

Step-indexing No liveness

→

Syntax for SL props

Later-free

No step-indexing ✔ **Liveness** ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Semantic alteration

Novel general approach §4

Invariant — Simple propositional sharing

Invariant — Simple propositional sharing

Propositional sharing

Sharing with contract by SL props

Modern approach to shared mutable state in SL ✻

Invariant — Simple propositional sharing

Propositional sharing

Sharing with contract by SL props

Modern approach to shared mutable state in SL

Invariant *Established by (Jung+ '15)*

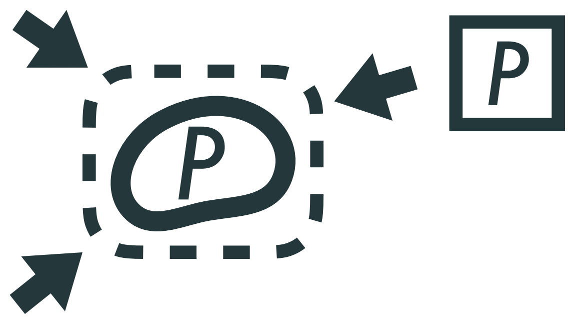
$\boxed{P}$ **Situation P always holds**

Share $\boxed{P} = \boxed{P} * \boxed{P}$

Cf. $\ell \mapsto v \neq \ell \mapsto v * \ell \mapsto v$

Imaginary store

Globally shared in verification



Invariant — Simple propositional sharing

Propositional sharing

Sharing with contract by SL props

Modern approach to shared mutable state in SL ✻

Invariant

Established by (Jung+ '15)

$\boxed{P}$ **Situation P always holds**

Share $\boxed{P} = \boxed{P} * \boxed{P}$

Cf. $\ell \mapsto v \neq \ell \mapsto v * \ell \mapsto v$

Imaginary store

Globally shared in verification

$\boxed{P}$

Example **Shared mutable ref**

ℓ: *ref bool*

ℓ

bool

$\boxed{\ell \mapsto \text{true} \lor \ell \mapsto \text{false}}$

$\{\ell \mapsto \text{true}\}\ \text{skip}\ \{\boxed{\ell \mapsto \text{true} \lor \ell \mapsto \text{false}}\}$ *Allocate*

$\{\boxed{\ell \mapsto \text{true} \lor \ell \mapsto \text{false}}\}\ \ell \leftarrow \text{false}\ \{\top\}$ *Store*

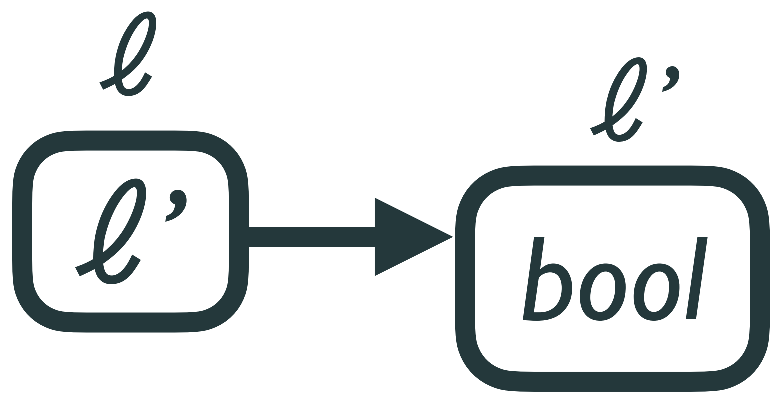
$\{\boxed{\ell \mapsto \text{true} \lor \ell \mapsto \text{false}}\}\ !\ell\ \{\lambda v.\ v = \text{true} \lor v = \text{false}\}$ *Load*

More invariant examples

More invariant examples

Example **Nested ref**

ℓ: ref (ref bool)



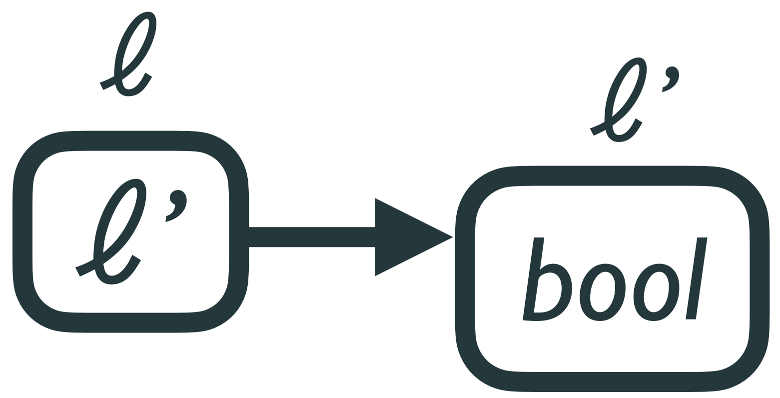
$$\boxed{\exists \ell'.\ \ell \mapsto \ell' * \boxed{\ell' \mapsto \mathsf{true} \lor \ell' \mapsto \mathsf{false}}}$$

Nested invariant

More invariant examples

Example **Nested ref**

ℓ: ref (ref bool)

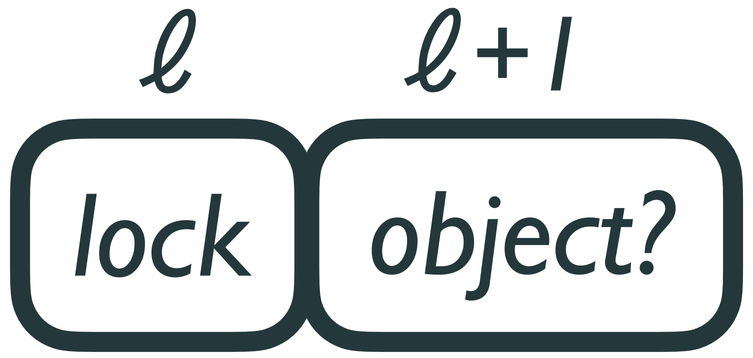


$$\boxed{\exists \ell'.\ \ell \mapsto \ell' * \boxed{\ell' \mapsto \text{true} \lor \ell' \mapsto \text{false}}}$$

Nested invariant

Example **Thread-safe ref to a mutex-guarded object**

ℓ: refmutex T



$$\boxed{\big(\ell \mapsto \text{false} * T\,(\ell+1)\big) \lor \ell \mapsto \text{true}}$$

Unlocked *Locked*

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic $*$

Old

Later modality $\triangleright$

Step-indexing No liveness

→

Syntax for SL props

Later-free

No step-indexing ✔ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Semantic alteration

Novel general approach §4

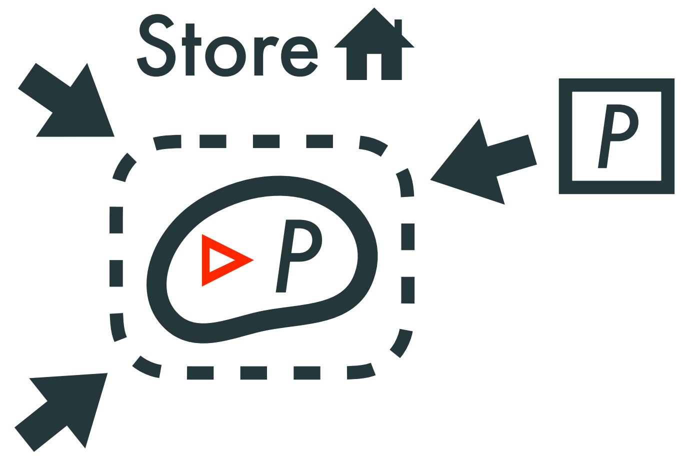
Old approach's problem: Later modality ⊳

Old approach's problem: Later modality ▷

Invariant access rule

$$\frac{\{\triangleright P * Q\}\ e\ \{\lambda v.\ \triangleright P * \Psi\, v\}}{\{\boxed{P} * Q\}\ e\ \{\Psi\}}$$

Weakened by
later modality ▷



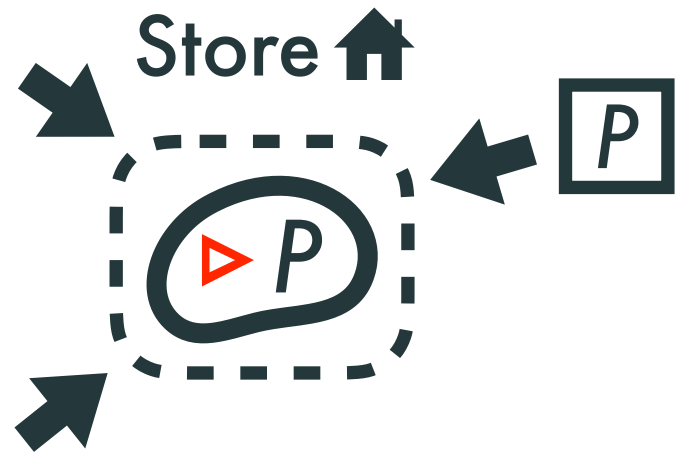
Naively store P not ▷P → Paradox!

Old approach's problem: Later modality ▷

Invariant access rule

$$\frac{\{\triangleright P * Q\}\ e\ \{\lambda v.\ \triangleright P * \Psi\, v\}}{\{\boxed{P} * Q\}\ e\ \{\Psi\}}$$

Weakened by
later modality ▷



Naively store P not ▷P → Paradox!

$$\triangleright\ \ell \mapsto v \ \equiv\ \ell \mapsto v$$

Under ◇

✔ $\boxed{\ell \mapsto \mathrm{true} \lor \ell \mapsto \mathrm{false}}$

Later in the way $\triangleright \boxed{P} \not\equiv \boxed{P}$

✘ $\boxed{\exists \ell'.\ \ell \mapsto \ell' * \boxed{\ell' \mapsto \mathrm{true} \lor \ell' \mapsto \mathrm{false}}}$

Nested invariant

Old workaround step-indexing & Its problem

Old workaround step-indexing & Its problem

Step-indexing

Laters stripped
as program executes

One execution step ↔ One later $\triangleright$

$$\frac{e \hookrightarrow e' \qquad \{P\}\ e'\ \{\Psi\}}{\{\triangleright P\}\ e\ \{\Psi\}}$$

Old workaround step-indexing & Its problem

Step-indexing

Laters stripped as program executes

One execution step ↔ One later ▷

$$\frac{e \hookrightarrow e' \qquad \{P\}\ e'\ \{\Psi\}}{\{{\color{red}\triangleright} P\}\ e\ \{\Psi\}}$$

cannot be used to verify liveness ♥

$$\cancel{\frac{e \hookrightarrow e' \qquad [P]\ e'\ [\Psi]}{[{\color{red}\triangleright} P]\ e\ [\Psi]}}$$

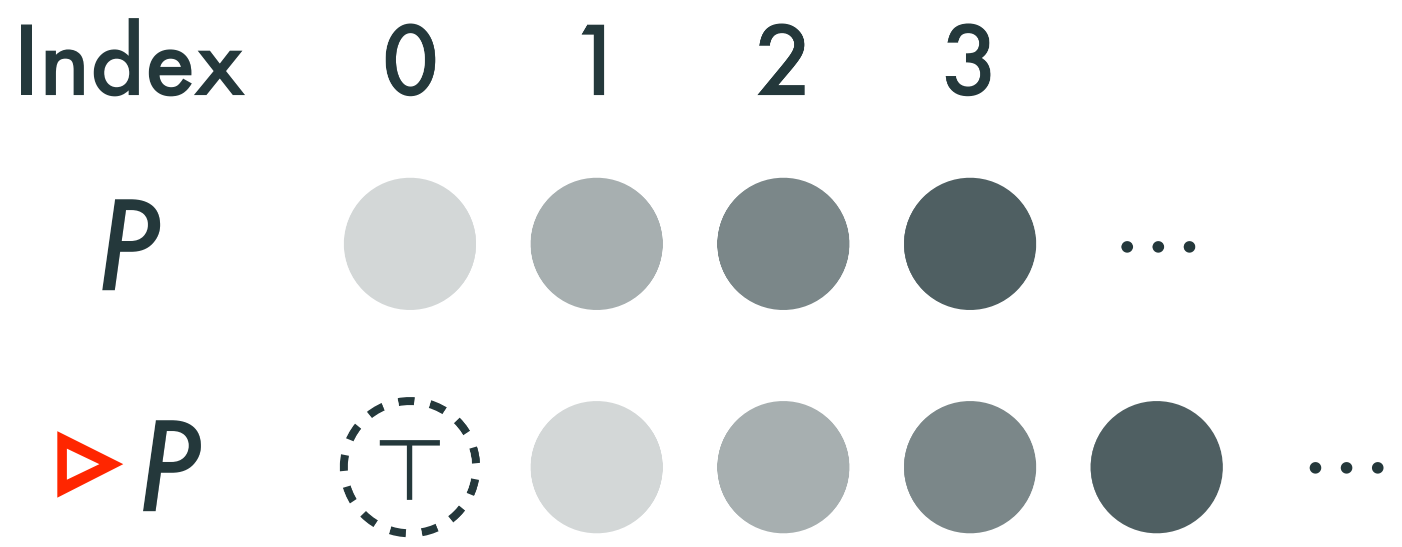
Termination guarantee lost

Where later ▷ comes from

Where later ▷ comes from

Indexing

Technique of semantics construction



Gradually define things as index grows

Later ▷ defers index by one

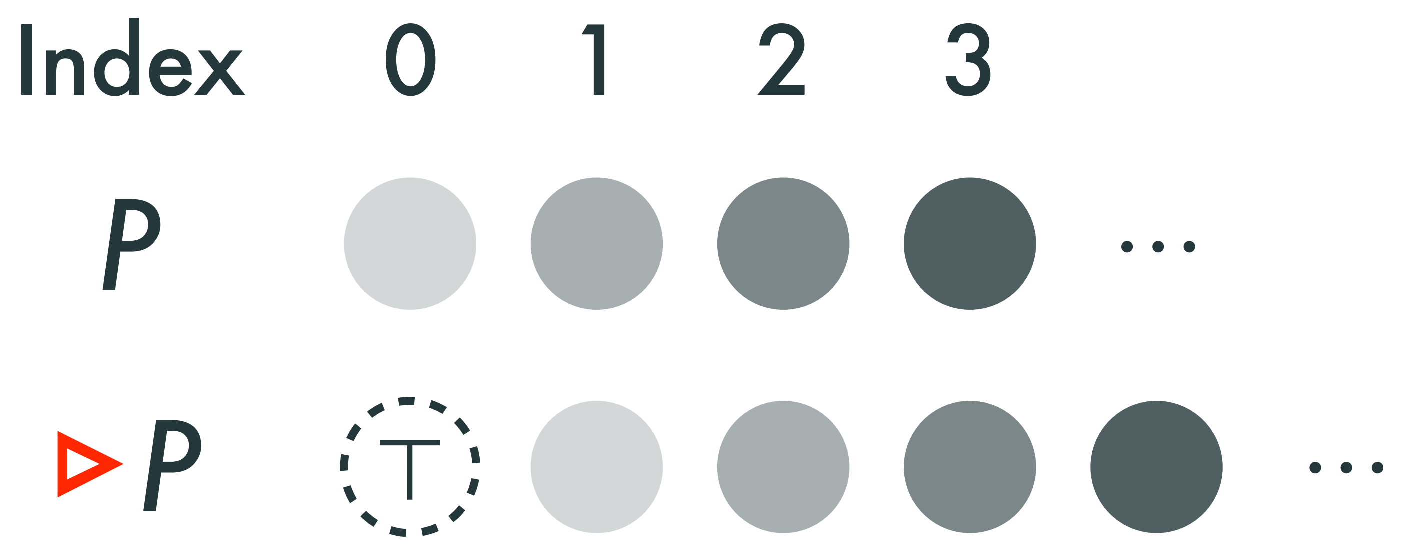
Non-idempotent

$\triangleright \triangleright P \neq \triangleright P$

Where later ▷ comes from

Indexing

Technique of semantics construction



Gradually define things as index grows

Later ▷ defers index by one

Non-idempotent

$\triangleright\triangleright P \neq \triangleright P$

For propositional sharing

Iris *(Jung+ '15) etc.*

✗ Ill-defined $\quad State \triangleq_? F\ iProp \qquad iProp \triangleq_? State \to Prop$

Defer by later $\quad State \triangleq F\ (\blacktriangleright iProp) \qquad iProp \triangleq State \to \widetilde{Prop}$

Later ▷ in store

Why later ⊳ damages liveness

Why later $\triangleright$ damages liveness

Model

Safety $\{P\}\, e\, \{\Psi\} \triangleq \Box(P \to \mathrm{wp}\ e\ \{\Psi\})$

$$\mathrm{wp}\ e\ \{\Psi\} \triangleq_{\nu} \cdots \lor \forall e' \hookleftarrow e.\ \mathrm{wp}\ e'\ \{\Psi\}$$

Coinductive Greatest fixpoint

Liveness ♥ $[P]\, e\, [\Psi] \triangleq \Box(P \to \mathrm{twp}\ e\ [\Psi])$

$$\mathrm{twp}\ e\ [\Psi] \triangleq_{\mu} \cdots \lor \forall e' \hookleftarrow e.\ \mathrm{twp}\ e'\ [\Psi]$$

Inductive Least fixpoint

Why later ▷ damages liveness

Model

Safety $\{P\}\, e\, \{\Psi\} \triangleq \Box\,(P \rightarrow \mathrm{wp}\; e\, \{\Psi\})$

$$\mathrm{wp}\; e \left\{\Psi\right\} \;\triangleq_{\nu}\; \cdots \vee \forall e' \hookleftarrow e.\; \mathrm{wp}\; e' \left\{\Psi\right\}$$

Coinductive Greatest fixpoint

Liveness ♥ $[P]\, e\, [\Psi] \triangleq \Box\,(P \rightarrow \mathrm{twp}\; e\, [\Psi])$

$$\mathrm{twp}\; e \left[\Psi\right] \;\triangleq_{\mu}\; \cdots \vee \forall e' \hookleftarrow e.\; \mathrm{twp}\; e' \left[\Psi\right]$$

Inductive Least fixpoint

Step-indexing

Safety ←

Later

$$\mathrm{wp}\; e \left\{\Psi\right\} \;\triangleq\; \cdots \vee \forall e' \hookleftarrow e.\; \triangleright\, \mathrm{wp}\; e' \left\{\Psi\right\}$$

Coinductive Guarded fixpoint

Index	0	1	2	3	
▷wp	$\top$				…

Why later ▷ damages liveness

Model

Safety $\{P\}\, e\, \{\Psi\} \triangleq \Box(P \to \mathrm{wp}\ e\ \{\Psi\})$

$$\mathrm{wp}\ e\ \{\Psi\} \ \triangleq_\nu\ \cdots \lor \forall e' \hookleftarrow e.\ \mathrm{wp}\ e'\ \{\Psi\}$$

Coinductive Greatest fixpoint

Liveness 💙 $[P]\, e\, [\Psi] \triangleq \Box(P \to \mathrm{twp}\ e\ [\Psi])$

$$\mathrm{twp}\ e\ [\Psi] \ \triangleq_\mu\ \cdots \lor \forall e' \hookleftarrow e.\ \mathrm{twp}\ e'\ [\Psi]$$

Inductive Least fixpoint

Step-indexing
Safety ←

Later

$$\mathrm{wp}\ e\ \{\Psi\} \ \triangleq\ \cdots \lor \forall e' \hookleftarrow e.\ \triangleright\, \mathrm{wp}\ e'\ \{\Psi\}$$

Coinductive **Guarded fixpoint**

Index	0	1	2	3	
▷wp	$\top$				…

Paradox

Step-indexing

$$\dfrac{\dfrac{\mathrm{loop} \hookrightarrow \mathrm{loop} \qquad \mathrm{twp\ loop}\ [\bot] \vDash \mathrm{twp\ loop}\ [\bot]}{\triangleright\, \mathrm{twp\ loop}\ [\bot] \vDash \mathrm{twp\ loop}\ [\bot]}}{\vDash \mathrm{twp\ loop}\ [\bot]}$$

Coinductivity by ▷ **Löb**

Non-termination 🐞

Big picture

Technical contributions

Core contribution of my work Recap

Separation logic ∗ *Scalable program logic for mutable state*

Verification goals	Basic SLs	Recent SLs *Iris (Jung+ '15) etc.*	**My work Nola** Framework for building SLs
Liveness ♥ Termination etc.	✔	✘	✔
		↑ Later modality ▷	↑ **Later-free** **Syntax for SL props**
Propositional Sharing	✘	✔	✔

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic ∗

Old			Syntax for SL props	
Later modality	$\triangleright$	➜	**Later-free**	
Step-indexing	No liveness		**No step-indexing**	**✔ Liveness ♥**

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic $*$

Old

Later modality $\triangleright$

Step-indexing No liveness

➜

Syntax for SL props

Later-free

No step-indexing ✔ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic ∗

Old

Later modality ▷

Step-indexing No liveness

➜

Syntax for SL props

Later-free

No step-indexing ✔ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic ∗

Old

Later modality ▷

Step-indexing No liveness

→

Syntax for SL props

Later-free

No step-indexing ✔ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic $*$

Old

Later modality $\triangleright$

Step-indexing No liveness

→

Syntax for SL props

Later-free

No step-indexing ✔ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic $*$

Old

Later modality $\triangleright$

Step-indexing No liveness

→

Syntax for SL props

Later-free

No step-indexing ✔ **Liveness** ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Semantic alteration

Novel general approach §4

Central topics

Invariant

Expressivity

Semantic alteration

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic ∗

Old

Later modality ▷

Step-indexing No liveness

→

Syntax for SL props

Later-free

No step-indexing ✓ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Semantic alteration

Novel general approach §4

Interface of Nola invariant

Interface of Nola invariant

Nola user

SL prop syntax

$nProp$

$nProp \ni P, Q ::=$
$\exists\,\Phi \mid P * Q \mid \ell \mapsto v \mid \cdots$

Nola library

Invariant

$\mathrm{inv}\,P \in iProp$
$P \in nProp$

Cf.

Old *Iris (Jung+ '15) etc.*

Invariant

$\boxed{P} \in iProp$
$P \in iProp$

Interface of Nola invariant

Nola user

SL prop syntax

$nProp$

$nProp \ni P, Q ::= \exists \Phi \mid P * Q \mid \ell \mapsto v \mid \cdots$

Interpretation

$[\![P]\!] \in iProp$

$\cdots \quad [\![P * Q]\!] \triangleq [\![P]\!] * [\![Q]\!] \quad \cdots$

Nola library

Invariant

$\mathrm{inv}\, P \in iProp$

$P \in nProp$

Later-free rules

$$\frac{\big[[\![P]\!] * Q\big]\ e\ \big[\lambda v.\ [\![P]\!] * \Psi\, v\big]'}{\big[\mathrm{inv}\, P * Q\big]\ e\ \big[\Psi\big]'}$$

✔ Liveness ♥

Cf.

Old *Iris (Jung+ '15) etc.*

Invariant

$\boxed{P} \in iProp$

$P \in iProp$

Rules with later

$$\frac{\big[\triangleright P * Q\big]\ e\ \big[\lambda v.\ \triangleright P * \Psi\, v\big]}{\big[\boxed{P} * Q\big]\ e\ \big[\Psi\big]}$$

Model for Nola invariant

Model for Nola invariant

My work Nola

Defer by syntax

$$State \triangleq F\ nProp \qquad iProp \triangleq State \longrightarrow Prop$$

$$\mathrm{inv}\,P \triangleq \exists\iota.\ \boxed{\circ[\iota := \mathrm{ag}\ P]}$$

Proposition itself

Old *Iris (Jung+ '15) etc.*

Defer by later

$$State \triangleq F\,(\blacktriangleright iProp) \qquad iProp \triangleq State \longrightarrow \widetilde{Prop}$$

$$\boxed{P} \triangleq \exists\iota.\ \boxed{\circ[\iota := \mathrm{ag}\ \mathrm{next}\ P]}$$

Equality weakened by later

Model for Nola invariant

My work Nola

Defer by syntax

$State \triangleq F\, nProp \quad iProp \triangleq State \longrightarrow Prop$

$\mathrm{inv}\, P \triangleq \exists \iota.\, \boxed{\circ[\iota := \mathrm{ag}\, P]}$

Proposition itself

$\mathrm{Winv}\, [\![\ ]\!] \triangleq \exists I.\ \cdots * \mathop{\huge *}_{\iota} (([\![I\, \iota]\!] * \cdots) \vee \cdots)$

Store **Later-free**

$[P]\, e\, [\Psi]' \triangleq [P]\, e\, [\Psi]^{\mathrm{Winv}\, [\![\ ]\!]}$ **Access the store**

No step-indexing → ✔ Liveness ♥

Old *Iris (Jung+ '15) etc.*

Defer by later

$State \triangleq F\, (\blacktriangleright iProp) \quad iProp \triangleq State \longrightarrow \widetilde{Prop}$

$\boxed{P} \triangleq \exists \iota.\, \boxed{\circ[\iota := \mathrm{ag}\, \mathrm{next}\, P]}$

Equality weakened by later

$\mathrm{Wiinv} \triangleq \exists I.\ \cdots * \mathop{\huge *}_{\iota} ((\triangleright I\, \iota * \cdots) \vee \cdots)$

Store Weakened by later

Step-indexing → No liveness

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic $*$

Old

Later modality $\triangleright$

Step-indexing No liveness

→

Syntax for SL props

Later-free

No step-indexing ✔ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Semantic alteration

Novel general approach §4

Verification target

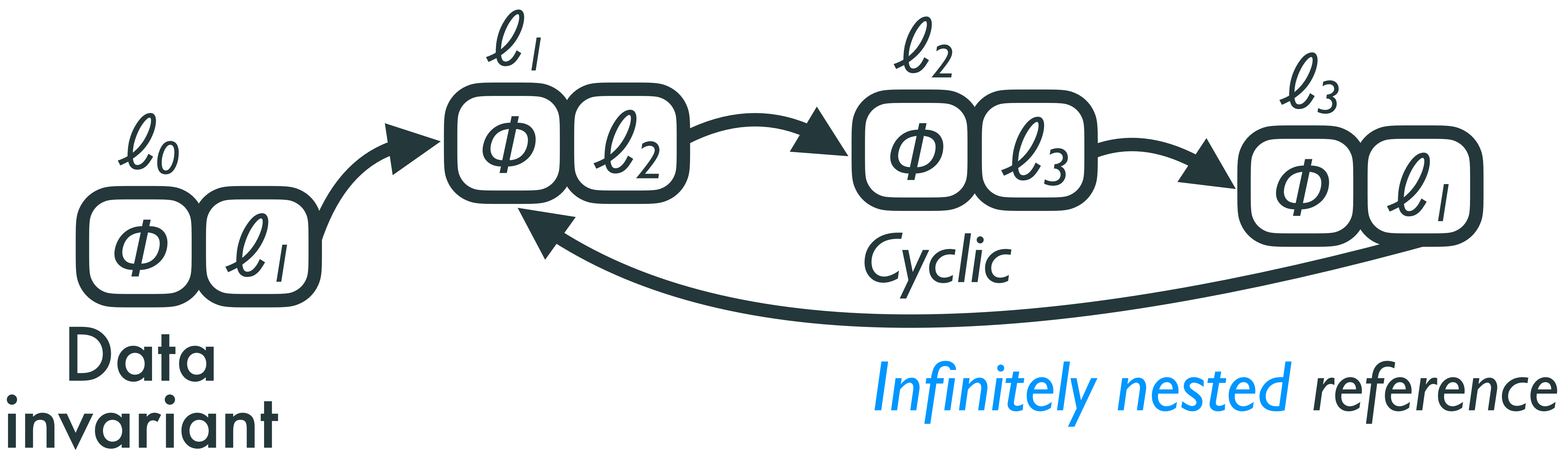
Verification target

Big goal **Verify total correctness ♥ on nested invariant**

Verification target

Big goal **Verify total correctness** ♥ **on nested invariant**

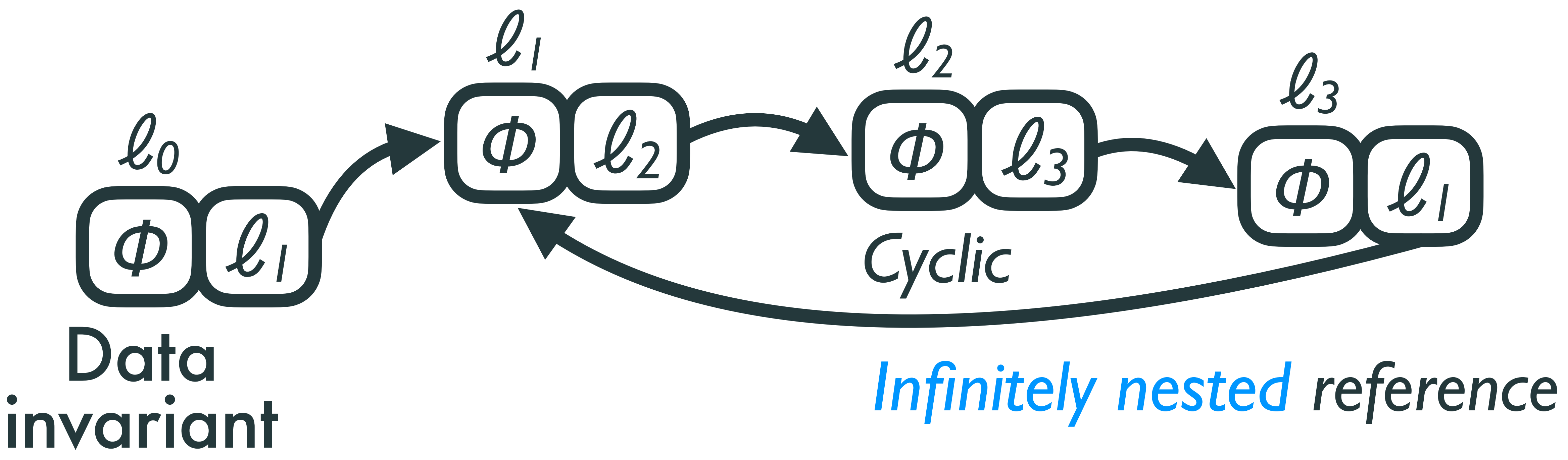
Data type **Shared mutable** singly linked list



Verification target

Big goal **Verify total correctness ♥ on nested invariant**

Data type **Shared mutable** singly linked list



Verify Iterative mutation $\text{fun iter}(\ell)\ \{\ \text{if } !c \neq 0\ \{\ f(\ell);\ c \leftarrow !c - 1;\ \text{iter}(!(\ell+1))\ \}\ \}$

over a list safely terminates ♥ if f safely terminates under **Φ**

E.g., $\text{fun } f(\ell)\ \{\ \ell \leftarrow !\ell + 3\ \}$ $\quad \Phi\ \lambda\ell.\ \exists k.\ \ell \mapsto 3k$

Verify termination with Nola invariant

Verify termination with Nola invariant

Construct SL prop syntax $\mathit{nProp} \ni P, Q ::=$

$$\exists\, \Phi \mid P * Q \mid \ell \mapsto v \mid \mathsf{inv}\, P \mid \mathtt{list}\ \Phi\ \ell \mid \cdots$$

Verify termination with Nola invariant

Construct SL prop syntax $\quad nProp \ni P, Q \ ::=$

$$\exists\, \Phi \mid P * Q \mid \ell \mapsto v \mid \mathsf{inv}\, P \mid \mathtt{list}\ \Phi\ \ell \mid \cdots$$

Construct semantic interpretation

$$[\![\exists\, \Phi]\!] \triangleq \exists a.\, [\![\Phi\ a]\!] \qquad [\![P * Q]\!] \triangleq [\![P]\!] * [\![Q]\!] \qquad [\![\ell \mapsto v]\!] \triangleq \ell \mapsto v \qquad [\![\mathsf{inv}\, P]\!] \triangleq \mathsf{inv}\, P$$

$$[\![\mathtt{list}\ \Phi\ \ell]\!] \triangleq \mathsf{inv}\,(\Phi\ \ell) * \mathsf{inv}\left(\exists\, \ell'.\ (\ell+1) \mapsto \ell' * \mathtt{list}\ \Phi\ \ell'\right)$$

Verify termination with Nola invariant

Construct SL prop syntax $\quad nProp \ni P, Q \ ::=$

$$\exists\, \Phi \mid P * Q \mid \ell \mapsto v \mid \mathrm{inv}\, P \mid \mathtt{list}\ \Phi\ \ell \mid \cdots$$

Construct semantic interpretation

$$[\![\exists\, \Phi]\!] \triangleq \exists a.\, [\![\Phi\ a]\!] \qquad [\![P * Q]\!] \triangleq [\![P]\!] * [\![Q]\!] \qquad [\![\ell \mapsto v]\!] \triangleq \ell \mapsto v \qquad [\![\mathrm{inv}\, P]\!] \triangleq \mathrm{inv}\, P$$

$$[\![\mathtt{list}\ \Phi\ \ell]\!] \triangleq \mathrm{inv}\,(\Phi\ \ell) * \mathrm{inv}\left(\exists\, \ell'.\, (\ell+1) \mapsto \ell' * \mathtt{list}\ \Phi\ \ell'\right)$$

Verify termination

$$\frac{\forall \ell.\ [\,\mathrm{inv}\,(\Phi\ \ell)\,]\ f(\ell)\ [\top]'}{[\,[\![\mathtt{list}\ \Phi\ \ell]\!] * c \mapsto n\,]\ \mathrm{iter}(\ell)\ [\,c \mapsto 0\,]'} \quad \because \textit{Induction over } n \in \mathbb{N}$$

$$[\,[\![\mathtt{list}\ \Phi\ \ell]\!]\,]\ !(\ell+1)\ [\lambda v.\ \exists \ell' = v.\ [\![\mathtt{list}\ \Phi\ \ell']\!]\,]'$$

Later-free access

Old

$$\mathrm{list}\ \Phi\ \ell \triangleq \boxed{\Phi\ \ell} * \boxed{\exists \ell'.\, (\ell+1) \mapsto \ell' * \mathrm{list}\ \Phi\ \ell'}$$

$$[\,\mathrm{list}\ \Phi\ \ell\,]\ !(\ell+1)\ [\lambda v.\ \exists \ell' = v.\ \triangleright\, \mathrm{list}\ \Phi\ \ell'\,]$$

Later

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic $*$

Old

Later modality $\triangleright$

Step-indexing No liveness

→

Syntax for SL props

Later-free

No step-indexing ✔ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Semantic alteration

Novel general approach §4

Analyze paradox in terms of Landin’s knot

Analyze paradox in terms of Landin's knot

Background   Shared mutable ref to a function causes **infinite loop**

Landin's knot

```
let r = ref id in
r := (λ _, !r ()); !r ()
```

Analyze paradox in terms of Landin's knot

Background   Shared mutable ref to a function causes **infinite loop**

Landin's knot

```
let r = ref id in
r := (λ _, !r ()); !r ()
```

r

Call !r

Self-reference

Paradox   *My contribution, a simplified version of (Krebbers+ '17+)'s*

Construct $\boxed{\boxed{S} \vee \Box \, {\vDash\!\!\Rrightarrow} \, \bot}$ $\rightarrow$ **Contradiction** ${\vDash} \, {\vDash\!\!\Rrightarrow} \, \bot$

Fancy update *"Logical function"*

Later-free invariant $P \; {\vDash} \, {\vDash\!\!\Rrightarrow} \; \boxed{P}$ $$\frac{P * Q \; {\vDash} \, {\vDash\!\!\Rrightarrow} \, (P * R)}{\boxed{P} * Q \; {\vDash} \, {\vDash\!\!\Rrightarrow} \; R}$$

Nola avoids paradoxes & subsumes the old way

Nola avoids paradoxes & subsumes the old way

Well-definedness of $[\![\ ]\!]$ naturally **avoids paradoxes**

$$\mathrm{inv}\ \mathrm{bad} \qquad [\![\mathrm{bad}]\!] \ \cancel{\triangleq}\,? \ \boxed{S} \lor \Box \Rrightarrow^{\mathrm{Winv}\,[\![\ ]\!]}_{\textbf{Store}} \bot$$

Fancy update

Ill-defined

Cyclic reference to $[\![\ ]\!]$

Nola avoids paradoxes & subsumes the old way

Well-definedness of $[\![\ ]\!]$ naturally avoids paradoxes

$\mathsf{inv}\ \mathsf{bad} \qquad [\![\mathsf{bad}]\!] \ \cancel{\triangleq}_? \ \boxed{S} \lor \square \Rrightarrow^{\mathrm{Winv}\,[\![\ ]\!]} \bot$

Store

Fancy update

Ill-defined

Cyclic reference to $[\![\ ]\!]$

$[\![\mathsf{thoare}\ P\ e\ \Psi]\!] \ \cancel{\triangleq}_? \ [\,[\![P]\!]\,]\ e\ [\,[\![\Psi]\!]\,]^{\mathrm{Winv}\,[\![\ ]\!]}$

Hoare triple

Internally uses **fancy update**

Avoid Landin's knot

Nola avoids paradoxes & subsumes the old way

Well-definedness of $[\![\ ]\!]$ naturally **avoids paradoxes**

$\mathrm{inv\ bad} \qquad [\![\mathrm{bad}]\!] \ \cancel{\triangleq}?\ \boxed{S} \lor \square \Rrightarrow^{\mathrm{Winv}\,[\![\ ]\!]}_{\text{Store}} \bot$

Fancy update

Ill-defined

Cyclic reference to $[\![\ ]\!]$

$[\![\mathrm{thoare}\ P\ e\ \Psi]\!] \ \cancel{\triangleq}?\ [\,[\![P]\!]\,]\ e\ [\,[\![\Psi]\!]\,]^{\mathrm{Winv}\,[\![\ ]\!]}$

Hoare triple

Internally uses **fancy update**

Avoid Landin's knot

Everything allowed under **later** → **Subsume the old way**

$[\![\triangleright\ \mathrm{hoare}\ P\ e\ \Psi]\!] \ \triangleq\ \triangleright \{\,[\![P]\!]\,\}\ e\ \{\,[\![\Psi]\!]\,\}^{\mathrm{Winv}\,[\![\ ]\!]}$

✔ Defer by later

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic ∗

Old

Later modality ▷

Step-indexing No liveness

→

Syntax for SL props

Later-free

No step-indexing ✔ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Semantic alteration

Novel general approach §4

Derivability for semantic alteration

Goal **Semantically alter props**

$$[\![\mathtt{inv}\,(P * Q)]\!] = [\![\mathtt{inv}\,(Q * P)]\!]$$

$$[\![\mathtt{inv}\ \mathtt{inv}\,(P * Q)]\!] = [\![\mathtt{inv}\ \mathtt{inv}\,(Q * P)]\!]$$

Derivability for semantic alteration

Goal **Semantically alter props**

$$[\![\mathtt{inv}\,(P * Q)]\!] = [\![\mathtt{inv}\,(Q * P)]\!]$$

$$[\![\mathtt{inv}\ \mathtt{inv}\,(P * Q)]\!] = [\![\mathtt{inv}\ \mathtt{inv}\,(Q * P)]\!]$$

Syntactic

✗ $[\![\mathrm{inv}\, P]\!] \triangleq \mathrm{inv}\, P$

Cyclic reference to $[\![\ ]\!]$

✗ $[\![\mathrm{inv}\, P]\!] \triangleq_{?} \exists Q.\ \Box\left([\![P]\!] \mathrel{*\!\!-\!\!*} [\![Q]\!]\right) * \mathrm{inv}\, Q$

Derivability for semantic alteration

Goal **Semantically alter props**

$$[\![\mathtt{inv}\,(P * Q)]\!] = [\![\mathtt{inv}\,(Q * P)]\!]$$
$$[\![\mathtt{inv}\ \mathtt{inv}\,(P * Q)]\!] = [\![\mathtt{inv}\ \mathtt{inv}\,(Q * P)]\!]$$

Syntactic

✗ $[\![\mathtt{inv}\,P]\!] \triangleq \mathtt{inv}\,P$

Cyclic reference to $[\![\ ]\!]$

✗ $[\![\mathtt{inv}\,P]\!] \triangleq_{?} \exists Q.\ \square\,([\![P]\!] \mathrel{*\!-\!*} [\![Q]\!]) * \mathtt{inv}\,Q$

Derivability

✔ $[\![\mathtt{inv}\,P]\!] \triangleq \exists Q.\ \square\,\mathrm{der}\,(P \mathrel{*\!-\!*} Q) * \mathtt{inv}\,Q$

Judgment

Sound $\mathrm{der}\,(P \mathrel{*\!-\!*} Q) \vDash [\![P]\!] \mathrel{*\!-\!*} [\![Q]\!]$ ➜ $\dfrac{[[\![P]\!] * Q]\ e\ [\lambda v.\ [\![P]\!] * \Psi v]'}{[[\![\mathtt{inv}\,P]\!] * Q]\ e\ [\Psi]'}$

Derivability for semantic alteration

Goal **Semantically alter props**

$$[\![\mathtt{inv}\,(P * Q)]\!] = [\![\mathtt{inv}\,(Q * P)]\!]$$

$$[\![\mathtt{inv}\ \mathtt{inv}\,(P * Q)]\!] = [\![\mathtt{inv}\ \mathtt{inv}\,(Q * P)]\!]$$

Syntactic

✗ $[\![\mathtt{inv}\,P]\!] \triangleq \mathtt{inv}\,P$

Cyclic reference to $[\![\ ]\!]$

✗ $[\![\mathtt{inv}\,P]\!] \triangleq_? \exists Q.\ \Box\,([\![P]\!] \ast\!\!-\!\!\ast [\![Q]\!]) * \mathtt{inv}\,Q$

Derivability

✔ $[\![\mathtt{inv}\,P]\!] \triangleq \exists Q.\ \Box\ \mathrm{der}\,(P \ast\!\!-\!\!\ast Q) * \mathtt{inv}\,Q$

Judgment

Sound $\mathrm{der}\,(P \ast\!\!-\!\!\ast Q) \vDash [\![P]\!] \ast\!\!-\!\!\ast [\![Q]\!]$ ➜ $\dfrac{[\,[\![P]\!] * Q\,]\ e\ [\lambda v.\ [\![P]\!] * \Psi v]'}{[\,[\![\mathtt{inv}\,P]\!] * Q\,]\ e\ [\Psi]'}$

Challenge **Construct sound & complete-ish derivability der**

Novel general approach to constructing der

Novel general approach to constructing der

Construct parameterized semantics

Defer by parameterization

$$[\![\mathtt{inv}\, P]\!]_\delta \triangleq \exists Q.\ \square\, \delta\,(P \ast\!\!-\!\!\ast\, Q) \ast \mathtt{inv}\, Q \qquad \cdots\ [\![P \ast Q]\!]_\delta \triangleq [\![P]\!]_\delta \ast [\![Q]\!]_\delta\ \cdots$$

Derivability candidate

Judgment semantics $[\![\ ]\!]^+ : (Judg \longrightarrow iProp) \longrightarrow (Judg \longrightarrow iProp)$ $\quad [\![P \ast\!\!-\!\!\ast\, Q]\!]^+_\delta \triangleq [\![P]\!]_\delta \ast\!\!-\!\!\ast [\![Q]\!]_\delta$

Novel general approach to constructing der

Construct parameterized semantics

Defer by parameterization

$$[\![\mathtt{inv}\, P]\!]_\delta \triangleq \exists Q.\ \Box\, \delta\, (P \mathrel{*\!\!-\!\!*} Q) * \mathtt{inv}\, Q \qquad \cdots\ [\![P * Q]\!]_\delta \triangleq [\![P]\!]_\delta * [\![Q]\!]_\delta\ \cdots$$

Derivability candidate

Judgment semantics $[\![\ ]\!]^+ : (Judg \to iProp) \to (Judg \to iProp)$ $\quad [\![P \mathrel{*\!\!-\!\!*} Q]\!]^+_\delta \triangleq [\![P]\!]_\delta \mathrel{*\!\!-\!\!*} [\![Q]\!]_\delta$

General der construction

$$\mathrm{der}\, J \triangleq_\mu \forall \delta \in \mathit{Deriv}\ \text{s.t.}\ \mathrm{der} \rightsquigarrow \delta.\ [\![J]\!]^+_\delta$$

Universally quantify semantics

$$\delta \in \mathit{Deriv} \triangleq_\mu \forall J.\ \big(\forall \delta' \in \mathit{Deriv}\ \text{s.t.}\ \delta \rightsquigarrow \delta'.\ [\![J]\!]^+_{\delta'}\big) \vDash \delta\, J$$

$$\delta \rightsquigarrow \delta' \triangleq \forall J.\ \Box\big(\delta\, J \to [\![J]\!]^+_{\delta'} \wedge \delta'\, J\big)$$

Novel general approach to constructing der

Construct parameterized semantics

Defer by parameterization

$$[\![\mathrm{inv}\, P]\!]_\delta \triangleq \exists Q.\ \Box\, \delta\,(P \ast\!-\!\ast Q) \ast \mathrm{inv}\, Q \qquad \cdots\ [\![P \ast Q]\!]_\delta \triangleq [\![P]\!]_\delta \ast [\![Q]\!]_\delta\ \cdots$$

Derivability candidate

Judgment semantics $[\![\ ]\!]^+ : (\mathit{Judg} \to \mathit{iProp}) \to (\mathit{Judg} \to \mathit{iProp})$ $\quad [\![P \ast\!-\!\ast Q]\!]^+_\delta \triangleq [\![P]\!]_\delta \ast\!-\!\ast [\![Q]\!]_\delta$

General der construction

$$\mathrm{der}\ J \triangleq_\mu \forall\, \delta \in \mathit{Deriv} \text{ s.t. } \mathrm{der} \rightsquigarrow \delta.\ [\![J]\!]^+_\delta$$

Universally quantify semantics

$$\delta \in \mathit{Deriv} \triangleq_\mu \forall J.\ (\forall\, \delta' \in \mathit{Deriv} \text{ s.t. } \delta \rightsquigarrow \delta'.\ [\![J]\!]^+_{\delta'}) \vDash \delta\, J$$

$$\delta \rightsquigarrow \delta' \triangleq \forall J.\ \Box\,(\delta\, J \to [\![J]\!]^+_{\delta'} \land \delta'\, J)$$

Member $\quad \mathrm{der} \in \mathit{Deriv} \quad \because$ *Definition*

Sound $\quad \mathrm{der}\ J \vDash [\![J]\!]^+_{\mathrm{der}} \quad \because$ *Induction*

Complete-ish *w.r.t. ∀ over Deriv*

$$(\forall\, \delta \in \mathit{Deriv}.\ \circledast\, \overline{[\![J']\!]^+_\delta} \twoheadrightarrow [\![J]\!]^+_\delta) \vDash \forall\, \delta \in \mathit{Deriv}.\ \circledast\, \overline{\delta\, J'} \twoheadrightarrow \delta\, J$$

etc.

Semantic alteration generally solved!

Other topics

Type soundness

Borrow

Prophetic borrow

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic *

Old

Later modality ▷

Step-indexing   No liveness

→

Syntax for SL props

Later-free

No step-indexing   ✔ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Semantic alteration

Novel general approach §4

Termination by leveled types verified with Nola

Termination by leveled types verified with Nola

Verification goal **Well-typed programs** **terminate ♥**

Termination by leveled types verified with Nola

Verification goal **Well-typed programs terminate ♥**

Leveled type system *Eliminate Landin's knot with levels $i \in \mathbb{N}$*

$T_i, U_i \ ::= \ \mathtt{ref}_k\, T_k \ \mid \ T_i \to_j U_i \ \ (j \le i) \ \mid \ \cdots$ Shared mutable ref & function

Restrict ref access $\dfrac{\Gamma \vdash e :_j \mathtt{ref}_i\, T \quad i < j}{\Gamma \vdash\, !e :_j T}$ $\dfrac{\Gamma \vdash e :_j \mathtt{ref}_i\, T \quad \Gamma \vdash e' :_j T \quad i < j}{\Gamma \vdash e \leftarrow e' :_j \mathtt{unit}}$

Termination by leveled types verified with Nola

Verification goal **Well-typed programs terminate ♥**

Leveled type system *Eliminate Landin's knot with levels $i \in \mathbb{N}$*

$$T_i, U_i \;::=\; \mathtt{ref}_k\, T_k \;\mid\; T_i \to_j U_i \;\; (j \le i) \;\mid\; \cdots$$

Shared mutable ref & function

Restrict ref access

$$\frac{\Gamma \vdash e :_j \mathtt{ref}_i\, T \quad i < j}{\Gamma \vdash\, !e :_j T} \qquad \frac{\Gamma \vdash e :_j \mathtt{ref}_i\, T \quad \Gamma \vdash e' :_j T \quad i < j}{\Gamma \vdash e \leftarrow e' :_j \mathtt{unit}}$$

Solution **Model type system with Nola invariants**

Semantic type judgment $[\![\overline{v : U} \vdash e :_i T]\!] \triangleq \left[\ast\, \overline{[\![U]\!]\, v} \right] e \left[[\![T]\!] \right]^{\ast_{k<i}\, \mathrm{Winv}\, [\![\;]\!]^*_k}$

$$[\![\mathtt{ref}\, T]\!]\, v \;\triangleq\; \exists \ell = v.\, \mathrm{inv}\,(\ell \mapsto T) \qquad [\![\ell \mapsto T]\!]^* \;\triangleq\; \exists w.\; \ell \mapsto w \ast [\![T]\!]\, w$$

$$[\![T \to_j U]\!]\, v \;\triangleq\; \forall u.\, \left[[\![T]\!]\, u \right] v(u) \left[[\![U]\!] \right]^{\ast_{k<j}\, \mathrm{Winv}\, [\![\;]\!]^*_k}$$

Construct interpretation by induction over the level

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic $*$

Old

Later modality $\triangleright$

Step-indexing No liveness

→

Syntax for SL props

Later-free

No step-indexing ✔ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Semantic alteration

Novel general approach §4

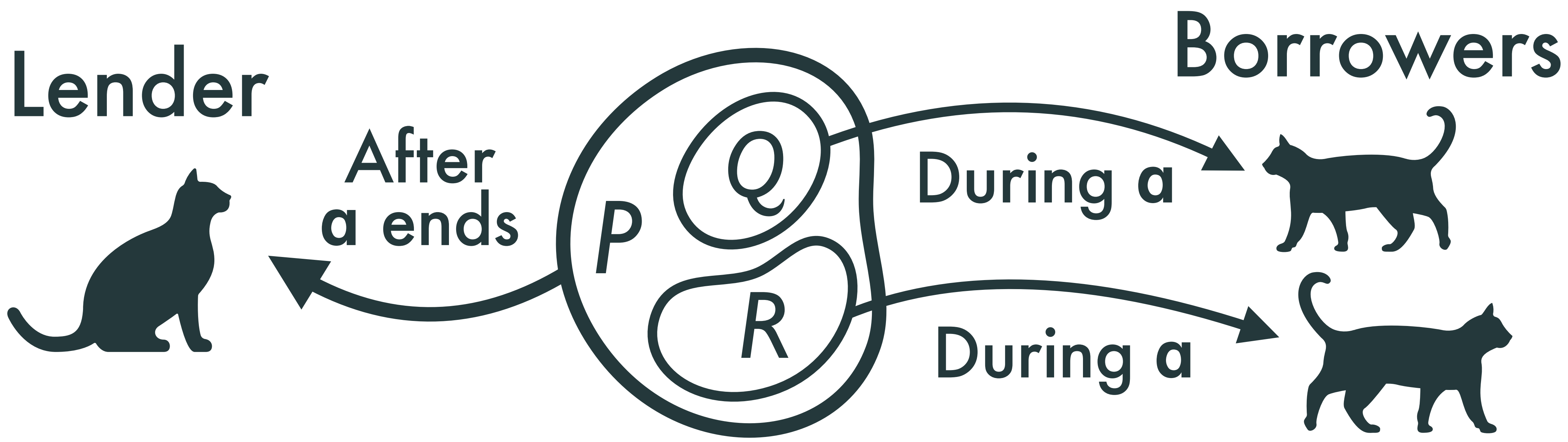
Rust-style borrow achieved in Nola

Rust-style borrow achieved in Nola

Rust-style borrow Lend / borrow ownership during a time period

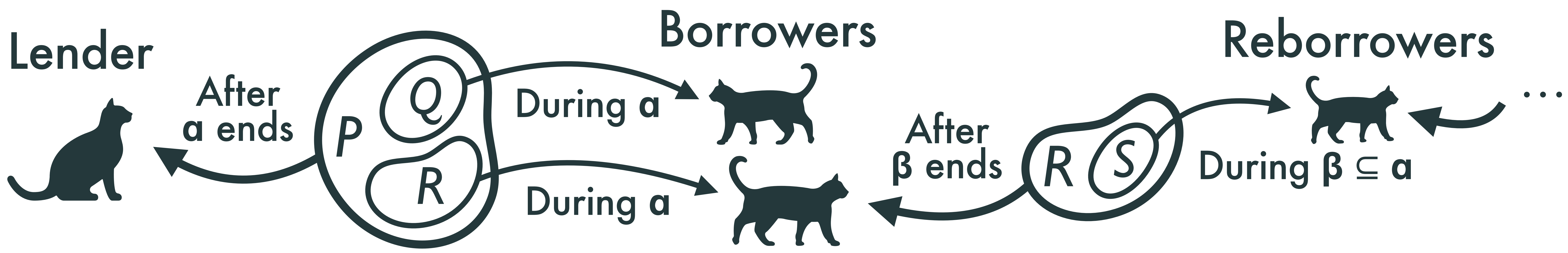
Rust-style borrow achieved in Nola

Rust-style borrow Lend / borrow ownership during a time period



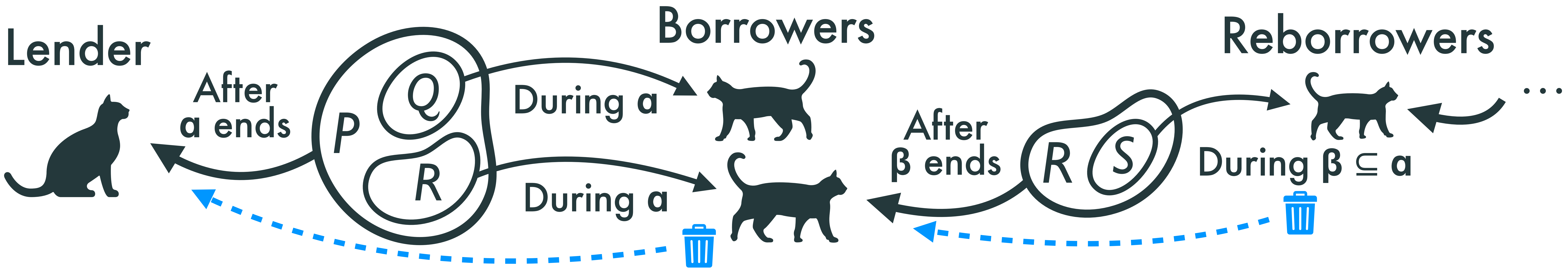
Rust-style borrow achieved in Nola

Rust-style borrow Lend / borrow ownership during a time period



Rust-style borrow achieved in Nola

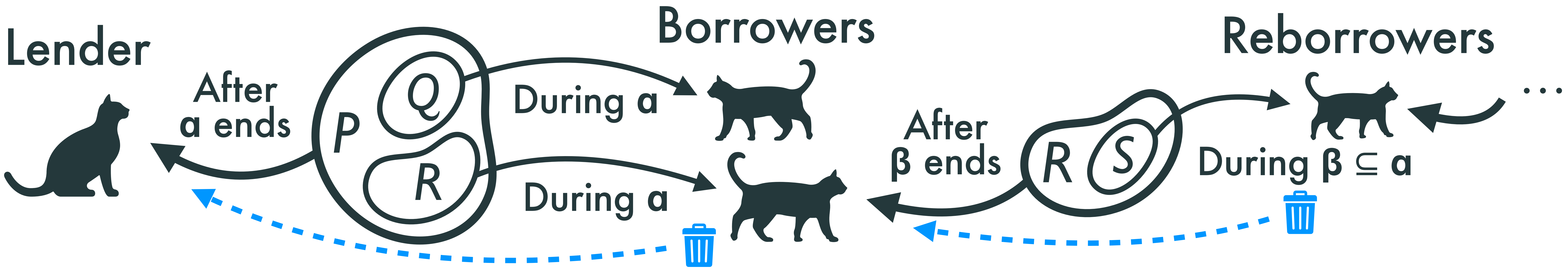
Rust-style borrow Lend / borrow ownership during a time period



Return ownership without direct communication ← Propositional sharing

Rust-style borrow achieved in Nola

Rust-style borrow Lend / borrow ownership during a time period



Return ownership without direct communication ← Propositional sharing

RustBelt *(Jung+ '18)*

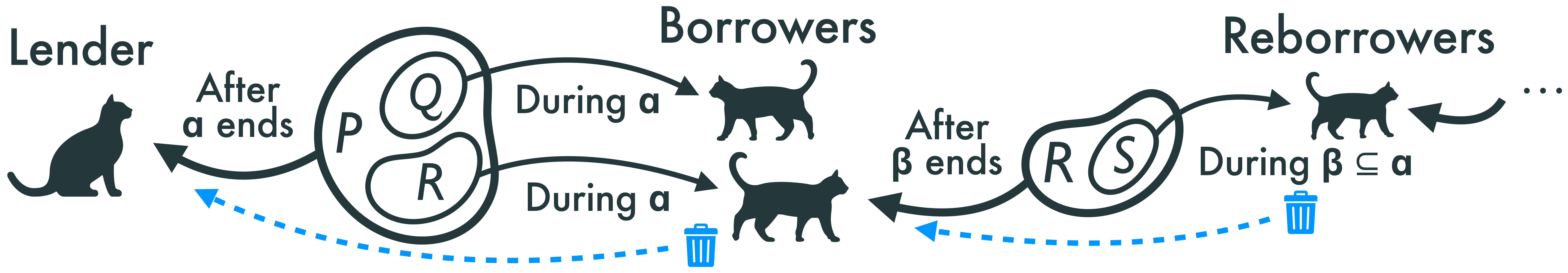
Borrows in SL to verify memory safety under Rust's ownership types

Later ⊳ → No liveness

Rust-style borrow achieved in Nola

Rust-style borrow

Lend / borrow ownership during a time period



Return ownership without direct communication ← Propositional sharing

RustBelt

(Jung+ '18)

Borrows in SL to verify memory safety under Rust's ownership types

Later ▷ → No liveness

Nola borrow

Later-free

Custom *nProp* & $[\![\ ]\!]$

Just like invariant

Rich operations

Subdivision, merger, reborrow

Proof rules

$$[\![P]\!] \vDash \Rrightarrow' \left(\mathrm{bor}^\alpha P \ * \ \mathrm{lend}^\alpha P\right)$$

$$[\alpha]_q \ * \ \mathrm{bor}^\alpha P \vDash \Rrightarrow' \left(\mathrm{obor}^\alpha_q P \ * \ [\![P]\!]\right)$$

$$[\alpha]_q \ * \ \mathrm{bor}^\alpha P \vDash \Rrightarrow' \left([\alpha]_q \ * \ \mathrm{bor}^{\alpha\sqcap\beta} P \ * \ (\dagger\alpha \twoheadrightarrow \mathrm{bor}^\alpha P)\right)$$

etc.

Technical contributions of my work Nola

Propositional sharing by syntax in separation logic ∗

Old

Later modality ▷

Step-indexing No liveness

➔

Syntax for SL props

Later-free

No step-indexing ✔ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

List mutation *Liveness × Nesting §3.3*

Type soundness *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Semantic alteration

Novel general approach §4

Key achievement: Nola prophetic borrow

Key achievement: Nola prophetic borrow

Background

RustHorn (Matsushita+ '20)

Verify functionally about borrows with **prophecy** of value at lifetime's end

Key achievement: Nola prophetic borrow

Background

RustHorn
(Matsushita+ ’20)

Verify functionally about borrows with **prophecy**
of value at lifetime’s end

RustHornBelt
(Matsushita+ ’22)

Later → No liveness

’s borrows × **Parametric prophecy**

RustHorn-style verification in SL Ad-hoc

Key achievement: Nola prophetic borrow

Background

RustHorn (Matsushita+ '20)

Verify functionally about borrows with **prophecy** of value at lifetime's end

RustHornBelt (Matsushita+ '22)

Later → No liveness

's borrows × **Parametric prophecy**

RustHorn-style verification in SL Ad-hoc

Nola prophetic borrow Nola borrow × 's parametric prophecy

Key achievement: Nola prophetic borrow

Background

RustHorn *(Matsushita+ '20)* Verify functionally about borrows with **prophecy** of value at lifetime's end

Later → No liveness

RustHornBelt *(Matsushita+ '22)* 's borrows × **Parametric prophecy**

RustHorn-style verification in SL Ad-hoc

Nola prophetic borrow

Nola borrow × 's parametric prophecy

Proof rules **Later-free** & **Abstract**

$$[\![\Phi\ a]\!] \ \vDash \Rrightarrow' \big(\exists x.\ \mathrm{bor}^{\alpha}_{a,x}\ \Phi \ * \ \mathrm{lend}^{\alpha}_{x}\ \Phi\big)$$

$$[\alpha]_q \ * \ \mathrm{bor}^{\alpha}_{a,x}\ \Phi \ \vDash \Rrightarrow' \big([\alpha]_q \ * \ \langle \lambda\pi.\ \pi\ x = a\rangle\big)$$

$$\dagger\alpha \ * \ \mathrm{lend}^{\alpha}_{x}\ \Phi \ \vDash \Rrightarrow' \big(\exists a.\ \langle \lambda\pi.\ \pi\ x = a\rangle * [\![\Phi\ a]\!]\big)$$

etc.

Key achievement: Nola prophetic borrow

Background

RustHorn (*Matsushita+ '20*)

Verify functionally about borrows with **prophecy** of value at lifetime's end

RustHornBelt (*Matsushita+ '22*)

Later → No liveness

's borrows × **Parametric prophecy**

RustHorn-style verification in SL Ad-hoc

Nola prophetic borrow

Nola borrow × 's parametric prophecy

Proof rules **Later-free** & **Abstract**

$$[\![\Phi\ a]\!] \ \vDash \Rrightarrow' \left(\exists x.\ \mathrm{bor}^{\alpha}_{a,x}\ \Phi \ * \ \mathrm{lend}^{\alpha}_{x}\ \Phi\right)$$

$$[\alpha]_q \ * \ \mathrm{bor}^{\alpha}_{a,x}\ \Phi \ \ \vDash \Rrightarrow' \left([\alpha]_q \ * \ \langle \lambda\pi.\, \pi\ x = a\rangle\right)$$

$$\dagger\alpha \ * \ \mathrm{lend}^{\alpha}_{x}\ \Phi \ \ \vDash \Rrightarrow' \left(\exists a.\ \langle \lambda\pi.\, \pi\ x = a\rangle * [\![\Phi\ a]\!]\right)$$

etc.

Model **Instantiate Nola borrow**

Internal custom syntax & interpretation

$$P^* \ ::= \ \mathtt{xbor}^{\gamma}_{x}\ \Phi \ \mid \ \mathtt{xlend}_{x}\ \Phi \ \mid \ \cdots$$

$$[\![\mathtt{xbor}^{\gamma}_{x}\ \Phi]\!]^* \ \triangleq \ \exists a.\ \mathrm{pc}^{\gamma}_{x}\ a * [\![\Phi\ a]\!] \quad \cdots$$

Closing

Related work

Summary

Related work

Related work

- **'Later-free' invariants in separation logic**
  - SteelCore *(Swamy+ '20)*, Later Credit *(Spies+ '22)*
    - *Still step-indexed & hiding laters → Liveness unsupported*
  - iCAP *(Svendsen+ '14)*, HOCAP *(Svendsen+ '13)*
    - *Nesting unsupported*

Related work

- **'Later-free' invariants in separation logic**
  - SteelCore *(Swamy+ '20)*, Later Credit *(Spies+ '22)*
    - *Still step-indexed & hiding laters → Liveness unsupported*
  - iCAP *(Svendsen+ '14)*, HOCAP *(Svendsen+ '13)*
    - *Nesting unsupported*
- **Liveness in step-indexed separation logic**
  - Transfinite Iris *(Spies+ '21)* — Indexing by ordinals
    - *Loses rules for later → Borrow unsupported*
    - *Requires bounding by ordinals etc. & Concurrency unsupported*

Summary — My work Nola

Technical contributions

Propositional sharing by syntax for SL props

Later-free → No step-indexing → ✔ Liveness ♥

Invariant *Simple & Powerful §3.2*

Case studies

- **List mutation** *Liveness × Nesting §3.3*
- **Type soundness** *Scalable & Flexible §5*

Borrow *Advanced & Foundation for Rust §6*

Prophetic borrow *Functionally verify §7*

Expressivity

What is paradoxical &
What can be shared §3.4

Semantic alteration

Novel general approach §4

High-level Mechanization Future applications Related work

Background Liveness Separation logic Old invariant Later