Later Credits: Reducing the Proof Cost of Step-Indexing in Iris

David Läwen
Advised by Robbert Krebbers

MFoCS Seminar

January 24th, 2023

Introduction

- Later Credits: A new technique for program verification in the context of *step-indexed separation logics*

Later Credits: Resourceful Reasoning for the Later Modality

SIMON SPIES, MPI-SWS, Saarland Informatics Campus, Germany
LENNARD GÄHER, MPI-SWS, Saarland Informatics Campus, Germany
JOSEPH TASSAROTTI, New York University, USA
RALF JUNG, MIT CSAIL, USA
ROBBERT KREBBERS, Radboud University Nijmegen, The Netherlands
LARS BIRKEDAL, Aarhus University, Denmark
DEREK DREYER, MPI-SWS, Saarland Informatics Campus, Germany

In the past two decades, step-indexed logical relations and separation logics have both come to play a major role in semantics and verification research. More recently, they have been married together in the form of *step-indexed separation logics* like VST, iCAP, and Iris, which provide powerful tools for (among other things) building semantic models of richly typed languages like Rust. In these logics, propositions are given semantics using a step-indexed model, and step-indexed reasoning is reflected into the logic through the so-called "later" modality. On the one hand, this modality provides an elegant, high-level account of step-indexed reasoning; on the other hand, when used in sufficiently sophisticated ways, it can become a nuisance, turning perfectly natural proof strategies into dead ends.

In this work, we introduce *later credits*, a new technique for escaping later-modality quagmires. By leveraging the second ancestor of these logics—separation logic—later credits turn "the right to eliminate a later" into an ownable resource, which is subject to all the traditional modular reasoning principles of separation logic. We develop the theory of later credits in the context of Iris, and present several challenging examples of proofs and proof patterns which were previously not possible in Iris but are now possible due to later credits.

100

CCS Concepts: • **Theory of computation** → **Separation logic**; **Logic and verification**.

Additional Key Words and Phrases: Separation logic, Iris, step-indexing, later modality, transfinite

ACM Reference Format:

Contents

Contents

A Counter API

- API with methods for creating, incrementing and reading a counter:

```
new() := ref(0)

inc(p) :=
  let x = !p in
    p ← x + 1

get(p) := !p
```

A Counter API

- Some client code...

```
let c = new() in
  (inc(c) || inc(c))
  get(c)
```

- ...and the API code

```
new() := ref(0)

inc(p) :=
  let x = !p in
    p ← x + 1

get(p) := !p
```

A Counter API

- Some client code...

```
let c = new() in
  (inc(c) || inc(c))
  get(c)
```

- ...and the API code

```
new() := ref(0)

inc(p) :=
  let x = !p in
    p ← x + 1

get(p) := !p
```

- Question: What is the result of `get(c)` in the client code?

Making the API Thread-Safe

How can we make the API thread-safe?

Making the API Thread-Safe

How can we make the API thread-safe?

Using the compare-and-set primitive: **`CAS`**(p, x, y)

- Semantics: *If value pointed to by p is equal to x, write y to p and return* **`true`**, *otherwise return* **`false`**

Making the API Thread-Safe

How can we make the API thread-safe?

Using the compare-and-set primitive: **CAS**(p, x, y)

- Semantics: *If value pointed to by* p *is equal to* x*, write* y *to* p *and return* **true***, otherwise return* **false**

```
new() := ref(0)

inc(p) :=
  let x = !p in
    if ...
    then ...
    else ...

get(p) := !p
```

Making the API Thread-Safe

How can we make the API thread-safe?

Using the compare-and-set primitive: **CAS**(p, x, y)

- Semantics: *If value pointed to by p is equal to x, write y to p and return* **true**, *otherwise return* **false**

```
new() := ref(0)

inc(p) :=
  let x = !p in
    if CAS(p, x, x + 1)
    then ()
    else inc(p)

get(p) := !p
```

Properties of the Counter API

```
let c = new() in
  (inc(c) || inc(c))
  get(c)
```

```
inc(p) :=
  let x = !p in
    if CAS(p, x, x + 1)
    then ()
    else inc(p)
```

The counter is now thread-safe:

- Calls to the API methods do not get stuck
- If calls to the API methods terminate, they yield the expected value

What have we seen so far?

- Assuming we wanted to verify thread-safety of the counter API:

```
inc(p) :=
  let x = !p in
    if CAS(p, x, x + 1)
    then ()
    else inc(p)
```

What have we seen so far?

- Assuming we wanted to verify thread-safety of the counter API:

```
inc(p) :=
  let x = !p in
    if CAS(p, x, x + 1)
    then ()
    else inc(p)
```

What do we need?	Verification tool

What have we seen so far?

- Assuming we wanted to verify thread-safety of the counter API:

```
inc(p) :=
  let x = !p in
    if CAS(p, x, x + 1)
    then ()
    else inc(p)
```

What do we need?	Verification tool
Reasoning about computation state	Hoare logic

What have we seen so far?

- Assuming we wanted to verify thread-safety of the counter API:

```
inc(p) :=
  let x = !p in
    if CAS(p, x, x + 1)
    then ()
    else inc(p)
```

What do we need?	**Verification tool**
Reasoning about computation state	Hoare logic
Modular reasoning about resources	Separation logic

What have we seen so far?

- Assuming we wanted to verify thread-safety of the counter API:

```
inc(p) :=
  let x = !p in
    if CAS(p, x, x + 1)
    then ()
    else inc(p)
```

What do we need?	Verification tool
Reasoning about computation state	Hoare logic
Modular reasoning about resources	Separation logic
Reasoning about concurrency	*Concurrent* separation logic

What have we seen so far?

- Assuming we wanted to verify thread-safety of the counter API:

```
inc(p) :=
  let x = !p in
    if CAS(p, x, x + 1)
    then ()
    else inc(p)
```

What do we need?	**Verification tool**
Reasoning about computation state	Hoare logic
Modular reasoning about resources	Separation logic
Reasoning about concurrency	*Concurrent* separation logic
Non-structural recursion	Step-indexing

What is Iris?

Step-indexed, concurrent separation logic framework

- Implemented in Coq
- Modularity: Specifications are reusable and composable
- Language-independent (Rust, OCaml, Scala, Go, ...)

Some Background on Iris

- Before discussing later credits, we need some background on Iris:

Iris from the ground up

A modular foundation for higher-order concurrent separation logic

RALF JUNG
MPI-SWS, Germany
(*e-mail:* jung@mpi-sws.org)

ROBBERT KREBBERS
Delft University of Technology, The Netherlands
(*e-mail:* mail@robbertkrebbers.nl)

JACQUES-HENRI JOURDAN
MPI-SWS, Germany
(*e-mail:* jjourdan@mpi-sws.org)

ALEŠ BIZJAK
Aarhus University, Denmark
(*e-mail:* abizjak@cs.au.dk)

LARS BIRKEDAL
Aarhus University, Denmark
(*e-mail:* birkedal@cs.au.dk)

DEREK DREYER
MPI-SWS, Germany
(*e-mail:* dreyer@mpi-sws.org)

Abstract

Iris is a framework for higher-order concurrent separation logic, which has been implemented in the Coq proof assistant and deployed very effectively in a wide variety of verification projects. Iris was designed with the express goal of simplifying and consolidating the foundations of modern separation logics, but it has evolved over time, and the design and semantic foundations of Iris itself have yet to be fully written down and explained together properly in one place. Here, we attempt to fill this gap, presenting a reasonably complete picture of the latest version of Iris (version 3.1), from first principles and in one coherent narrative.

Propositions in Iris

P, Q	$::=$	$\mathsf{False} \quad \mid \quad \mathsf{True}$	*Boolean values*
	$\mid$	$\ldots$	
	$\mid$	$P \lor Q$	*Disjunction*
	$\mid$	$\exists x : \tau.\ P$	*Ex. quantification*
	$\mid$	$\forall x : \tau.\ P$	*Univ. quantification*
	$\mid$	$\{P\}\ e\ \{v.\ Q\}$	*Hoare triples*
	$\mid$	$P * Q$	*Separating conjunction*
	$\mid$	$\ell \mapsto v$	*Points-to assertion*
	$\mid$	$\triangleright P$	*Later modality*
	$\mid$	$\ldots$	

Propositions in Iris

$$
\begin{array}{rcll}
P, Q & ::= & \mathsf{False} \mid \mathsf{True} & \textit{Boolean values} \\
& \mid & \ldots & \\
& \mid & P \vee Q & \textit{Disjunction} \\
& \mid & \exists x : \tau.\ P & \textit{Ex. quantification} \\
& \mid & \forall x : \tau.\ P & \textit{Univ. quantification} \\
& \mid & \{P\}\ e\ \{v.\ Q\} & \textit{Hoare triples} \\
& \mid & P * Q & \textit{Separating conjunction} \\
& \mid & \ell \mapsto v & \textit{Points-to assertion} \\
& \mid & \triangleright P & \textit{Later modality} \\
& \mid & \ldots &
\end{array}
$$

- Higher-order logic: Quantifiers can range over any type
- Hoare triples are first-order

Hoare Logic

Hoare triples express partial program correctness:

$$\{P\}\ e\ \{v.\ Q\}$$

Hoare Logic

Hoare triples express partial program correctness:

$$\{P\}\ e\ \{v.\ Q\}$$

For an initial state satisfying *precondition* P:

- Execution of e does not crash
- If e terminates with value v', the final state satisfies the *postcondition* Q by $Q[v \leftarrow v']$

Separation Logic

- Propositions convey *ownership* of state

Separation Logic

- Propositions convey *ownership* of state
- Points-to assertion: $\ell \mapsto v$

Separation Logic

- Propositions convey *ownership* of state
- Points-to assertion: $\ell \mapsto v$
- Separating conjunction: $P * Q$

Separation Logic

- Propositions convey *ownership* of state
- Points-to assertion: $\ell \mapsto v$
- Separating conjunction: $P * Q$
  - $(\ell \mapsto v) * (k \mapsto w)$ implies $\ell \neq k$

Separation Logic

- Propositions convey *ownership* of state
- Points-to assertion: $\ell \mapsto v$
- Separating conjunction: $P * Q$
  - $(\ell \mapsto v) * (k \mapsto w)$ implies $\ell \neq k$
- Frame rule:

$$\textsc{Frame}\quad \frac{\{P\}\ e\ \{v.\ Q\}}{\{P * R\}\ e\ \{v.\ (Q * R)\}}$$

Concurrent Separation Logic

- Parallel composition rule of CSL:

$$\textsc{Par}\quad \frac{\{P_1\}\; e_1\; \{v.\; Q_1\} \qquad \{P_2\}\; e_2\; \{v.\; Q_2\}}{\{P_1 * P_2\}\;\; e_1 \,||\, e_2\;\; \{v.\; Q_1 * Q_2\}}$$

Contents

Contents

Step-Indexing

Technique for reasoning about (languages with) “cyclic” features

- E.g. higher-order state:

$$
\begin{aligned}
&r \leftarrow \lambda x.\, (!\, r)\ x\,; \\
&(!\, r)\ ()
\end{aligned}
$$

Step-Indexing

Technique for reasoning about (languages with) "cyclic" features

- E.g. higher-order state:

$$r \leftarrow \lambda x.\ (!\,r)\ x;$$
$$(!\,r)\ ()$$

- Problem: Naive models are unsound

Step-Indexing

Technique for reasoning about (languages with) “cyclic” features

- E.g. higher-order state:

$$r \leftarrow \lambda x.\ (!\,r)\ x;$$
$$(!\,r)\ ()$$

- Problem: Naive models are unsound
- Solution: *Stratify* the model with step-indices
  - Expression e satisfies predicate P with n steps “on the clock” if, when e reduces to e' in $i < n$ steps, then e' satisfies P with $n - i$ steps left “on the clock”

A Minimal Step-Indexed Logic

- Expression grammar:

$$
\begin{array}{rclclr}
P, Q & ::= & \mathsf{False} & \mid & \mathsf{True} & \textit{Boolean values} \\
 & \mid & P \wedge Q & & & \textit{Conjunction} \\
 & \mid & P \vee Q & & & \textit{Disjunction} \\
 & \mid & P \Rightarrow Q & & & \textit{Implication} \\
 & \mid & \exists x.\ P & & & \textit{Ex. quantification} \\
 & \mid & \forall x.\ P & & & \textit{Univ. quantification}
\end{array}
$$

A Minimal Step-Indexed Logic

- Semantic model: Step-indexed propositions are interpreted as downwards-closed subsets of $\mathbb{N}$.

A Minimal Step-Indexed Logic

- Semantic model: Step-indexed propositions are interpreted as downwards-closed subsets of $\mathbb{N}$.

$$SProp \triangleq \{A \subseteq \mathbb{N} \mid \forall n \geq m.\ \ n \in A \Rightarrow m \in A\} = \mathcal{P}^{\downarrow}(\mathbb{N})$$

$$[\![-]\!] : Prop \longrightarrow SProp$$

A Minimal Step-Indexed Logic

- Semantic model: Step-indexed propositions are interpreted as downwards-closed subsets of $\mathbb{N}$.

$$SProp \triangleq \{A \subseteq \mathbb{N} \mid \forall n \geq m.\ \ n \in A \Rightarrow m \in A\} = \mathcal{P}^{\downarrow}(\mathbb{N})$$

$$[\![-]\!] : Prop \longrightarrow SProp$$

$$[\![\text{False}]\!] \triangleq \emptyset \qquad [\![P \wedge Q]\!] \triangleq [\![P]\!] \cap [\![Q]\!]$$

$$[\![\text{True}]\!] \triangleq \mathbb{N} \qquad [\![P \vee Q]\!] \triangleq [\![P]\!] \cup [\![Q]\!]$$

$$[\![P \Rightarrow Q]\!] \triangleq \{n \mid \forall m \leq n.\ \ m \in [\![P]\!] \Rightarrow m \in [\![Q]\!]\}$$

A Minimal Step-Indexed Logic

- Semantic model: Step-indexed propositions are interpreted as downwards-closed subsets of $\mathbb{N}$.

$$SProp \triangleq \{A \subseteq \mathbb{N} \mid \forall n \geq m.\ \ n \in A \Rightarrow m \in A\} = \mathcal{P}^{\downarrow}(\mathbb{N})$$

$$[\![-]\!] : Prop \longrightarrow SProp$$

$$[\![\mathsf{False}]\!]_\rho \triangleq \emptyset \qquad [\![P \wedge Q]\!]_\rho \triangleq [\![P]\!]_\rho \cap [\![Q]\!]_\rho$$

$$[\![\mathsf{True}]\!]_\rho \triangleq \mathbb{N} \qquad [\![P \vee Q]\!]_\rho \triangleq [\![P]\!]_\rho \cup [\![Q]\!]_\rho$$

$$[\![P \Rightarrow Q]\!]_\rho \triangleq \{n \mid \forall m \leq n.\ \ m \in [\![P]\!]_\rho \Rightarrow m \in [\![Q]\!]_\rho\}$$

$$[\![\forall x.P]\!]_\rho \triangleq \bigcap_v [\![P]\!]_{\rho[x \leftarrow v]} \qquad [\![x]\!]_\rho \triangleq \rho(x)$$

$$[\![\exists x.P]\!]_\rho \triangleq \bigcup_v [\![P]\!]_{\rho[x \leftarrow v]}$$

A Minimal Step-Indexed Logic

- A selection of rules:

$$P \vdash \text{True} \qquad \text{False} \vdash P$$

$$\frac{P \qquad Q}{P \wedge Q} \qquad P \wedge Q \vdash P \qquad P \wedge Q \vdash Q$$

$$P \vdash P \vee Q \qquad Q \vdash P \vee Q \qquad \frac{P \vdash R \qquad Q \vdash R}{P \vee Q \vdash R}$$

- Nothing new or exciting so far...

The Later Modality

- Updated expression grammar:

P, Q	$::=$	False | True		*Boolean values*
	|	$P \wedge Q$		*Conjunction*
	|	$P \vee Q$		*Disjunction*
	|	$P \Rightarrow Q$		*Implication*
	|	$\exists x : \tau.\ P$		*Ex. quantification*
	|	$\forall x : \tau.\ P$		*Univ. quantification*
	|	$\triangleright P$		*Later modality*

The Later Modality

- Updated semantic model:

$$SProp \triangleq \{A \subseteq \mathbb{N} \mid \forall n \geq m.\ \ n \in A \Rightarrow m \in A\}$$

$$[\![-]\!] : Prop \longrightarrow SProp$$

$$[\![\mathsf{False}]\!]_\rho \triangleq \emptyset \qquad [\![P \wedge Q]\!]_\rho \triangleq [\![P]\!]_\rho \cap [\![Q]\!]_\rho$$

$$[\![\mathsf{True}]\!]_\rho \triangleq \mathbb{N} \qquad [\![P \vee Q]\!]_\rho \triangleq [\![P]\!]_\rho \cup [\![Q]\!]_\rho$$

$$[\![P \Rightarrow Q]\!]_\rho \triangleq \{n \mid \forall m \leq n.\ \ m \in [\![P]\!]_\rho \Rightarrow m \in [\![Q]\!]_\rho\}$$

$$[\![\forall x.P]\!]_\rho \triangleq \bigcap_v [\![P]\!]_{\rho[x \leftarrow v]} \qquad [\![x]\!]_\rho \triangleq \rho(x)$$

$$[\![\exists x.P]\!]_\rho \triangleq \bigcup_v [\![P]\!]_{\rho[x \leftarrow v]}$$

$$[\![\triangleright P]\!]_\rho \triangleq \{n \mid n = 0 \vee (n-1) \in [\![P]\!]_\rho\}$$

The Later Modality

- Rules:

LaterIntro

$$P \vdash \triangleright P$$

LaterMono

$$\frac{P \vdash Q}{\triangleright P \vdash \triangleright Q}$$

Löb

$$(\triangleright P \Rightarrow P) \vdash P$$

Later Elimination

- The Later Elimination Problem:
  $\triangleright P$ is in the context, but P is necessary to proceed,
  i.e. the guarding later must be *eliminated*

Later Elimination

- The Later Elimination Problem:
  $\triangleright P$ is in the context, but P is necessary to proceed,
  i.e. the guarding later must be *eliminated*

- Three common techniques:
  - Timelessness of propositions

Later Elimination

- The Later Elimination Problem:
  $\triangleright P$ is in the context, but P is necessary to proceed,
  i.e. the guarding later must be *eliminated*
- Three common techniques:
  - Timelessness of propositions

If P is timeless, a later modality guarding P can be “stripped away”.
A proposition P is timeless, *iff*

$$\forall n.\ \ 0 \in [\![P]\!] \Rightarrow n \in [\![P]\!].$$

Later Elimination

- The Later Elimination Problem:
  $\triangleright P$ is in the context, but P is necessary to proceed,
  i.e. the guarding later must be *eliminated*

- Three common techniques:
  - Timelessness of propositions

If P is timeless, a later modality guarding P can be "stripped away".

$$\textsc{Timeless}\quad \frac{\{P * Q\}\ e\ \{v.\ R\} \qquad \text{timeless}(P)}{\{\triangleright P * Q\}\ e\ \{v.\ R\}}$$

Later Elimination

- The Later Elimination Problem:
  $\triangleright P$ is in the context, but P is necessary to proceed,
  i.e. the guarding later must be *eliminated*

- Three common techniques:
  - Timelessness of propositions
  - Commuting rules

LaterSep
$$\triangleright(P * Q) \dashv\vdash \triangleright P * \triangleright Q$$

LaterConj
$$\triangleright(P \vee Q) \dashv\vdash \triangleright P \vee \triangleright Q$$

LaterExist
$$\frac{\text{inhabited}(x)}{\triangleright(\exists x.\, \triangleright P) \dashv\vdash \exists x.\, \triangleright P}$$

Later Elimination

- The Later Elimination Problem:
  $\triangleright P$ is in the context, but P is necessary to proceed,
  i.e. the guarding later must be *eliminated*

- Three common techniques:
  - Timelessness of propositions
  - Commuting rules
  - Program steps (PureStep)

$$\textsc{PureStep}\quad \frac{\{P\}\ e'\ \{v.\ Q\} \qquad e \rightarrow_{\text{pure}} e'}{\{\triangleright P\}\ e\ \{v.\ Q\}}$$

Idea: Amortisation of Later Eliminations

- Instead of one later elimination per step...

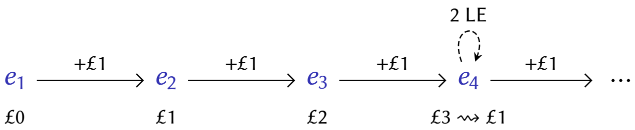
$$e_1 \xrightarrow{\text{1 LE}} e_2 \xrightarrow{\text{1 LE}} e_3 \xrightarrow{\text{1 LE}} e_4 \xrightarrow{\text{1 LE}} \cdots$$

Idea: Amortisation of Later Eliminations

- Instead of one later elimination per step...

$$e_1 \xrightarrow{\text{1 LE}} e_2 \xrightarrow{\text{1 LE}} e_3 \xrightarrow{\text{1 LE}} e_4 \xrightarrow{\text{1 LE}} \cdots$$

- ... generate one "token" per step which is redeemable for one later elimination, possibly at a *later* stage:



Contents

Contents

A Brief Interlude: Resources and the Update Modality

- *Resources* are a form of non-physical state, subject to the reasoning principles of separation logic

A Brief Interlude: Resources and the Update Modality

- *Resources* are a form of non-physical state, subject to the reasoning principles of separation logic
- Resources are updated according to resource-specific update rules
- Update modality ($\Rrightarrow$) allows reasoning about the result of updating resources
- $\Rrightarrow P$ states that P holds after applying updates to the resources

A Brief Interlude: Resources and the Update Modality

- *Resources* are a form of non-physical state, subject to the reasoning principles of separation logic
- Resources are updated according to resource-specific update rules
- Update modality ($\Rrightarrow$) allows reasoning about the result of updating resources
- $\Rrightarrow P$ states that P holds after applying updates to the resources

UpdIntro
$$P \vdash \Rrightarrow P$$

UpdMono
$$\frac{P \vdash Q}{\Rrightarrow P \vdash \Rrightarrow Q}$$

UpdTrans
$$\Rrightarrow \Rrightarrow P \vdash \Rrightarrow P$$

UpdFrame
$$P * \Rrightarrow Q \vdash \Rrightarrow (P * Q)$$

UpdExec
$$\frac{\{P\}\ e\ \{v.\ Q\}}{\{\Rrightarrow P\}\ e\ \{v.\ Q\}}$$

Later Credits

- Two main components of later credits in Iris:
  - New resource type $\pounds n$
  - New update modality ($\Rrightarrow_{\mathsf{le}}$) (*later elimination update*)

Later Credits

- Two main components of later credits in Iris:
  - New resource type $£n$
  - New update modality ($\Rrightarrow_{\mathsf{le}}$) (*later elimination update*)

- $£1$ represents the "right to eliminate" one later

Later Credits

- Two main components of later credits in Iris:
  - New resource type $£n$
  - New update modality ($\Rrightarrow_{\mathsf{le}}$) (*later elimination update*)

- $£1$ represents the “right to eliminate” one later

$$\textsc{PureStep}\ \frac{\{P\}\ e'\ \{v.\ Q\} \qquad e \rightarrow_{\mathsf{pure}} e'}{\{\triangleright P\}\ e\ \{v.\ Q\}}$$

Later Credits

- Two main components of later credits in Iris:
  - New resource type $\pounds n$
  - New update modality ($\Rrightarrow_{\mathsf{le}}$) (*later elimination update*)

- $\pounds 1$ represents the "right to eliminate" one later

$$\textsc{PureStep}\quad \frac{\{P * \pounds 1\}\; e' \;\{v.\, Q\} \qquad e \rightarrow_{\mathsf{pure}} e'}{\{P\}\; e \;\{v.\, Q\}}$$

$$\textsc{LEUpdLater}\quad \pounds 1 * \triangleright P \vdash \Rrightarrow_{\mathsf{le}} P$$

$$\textsc{LEUpdExec}\quad \frac{\{P\}\; e \;\{v.\, Q\}}{\{\Rrightarrow_{\mathsf{le}} P\}\; e \;\{v.\, Q\}}$$

Later Credits

- Two main components of later credits in Iris:
  - New resource type $£n$
  - New update modality ($\Rrightarrow_{\mathrm{le}}$) (*later elimination update*)

- $£1$ represents the "right to eliminate" one later

PureStep
$$\frac{\{P * £1\}\ e'\ \{v.\ Q\} \qquad e \rightarrow_{\mathrm{pure}} e'}{\{P\}\ e\ \{v.\ Q\}}$$

LEUpdLater
$$£1 * \triangleright P \vdash \Rrightarrow_{\mathrm{le}} P$$

LEUpdExec
$$\frac{\{P\}\ e\ \{v.\ Q\}}{\{\Rrightarrow_{\mathrm{le}} P\}\ e\ \{v.\ Q\}}$$

- Facilitates reasoning about later eliminations as an ownable resource

CreditSplit
$$£(n+m) \Leftrightarrow £n * £m$$

Propositions in Iris

P, Q	$::=$	$\mathsf{False} \quad \vert \quad \mathsf{True}$	*Boolean values*
	$\vert$	$\ldots$	
	$\vert$	$P \vee Q$	*Disjunction*
	$\vert$	$\exists x : \tau.\ P$	*Ex. quantification*
	$\vert$	$\forall x : \tau.\ P$	*Univ. quantification*
	$\vert$	$\{P\}\ e\ \{v.\ Q\}$	*Hoare triples*
	$\vert$	$P * Q$	*Separating conjunction*
	$\vert$	$\ell \mapsto v$	*Points-to connective*
	$\vert$	$\triangleright P$	*Later modality*
	$\vert$	$£n$	*Later credits*
	$\vert$	$\Rrightarrow_{\mathsf{le}} P$	*Later elimination update modality*
	$\vert$	$\ldots$	

Later Credits for Reordering Refinements

- Expression e contextually refines expression e', denoted by $e \leq_{\text{ctx}} e'$, if no well-typed context C can distinguish the two.

Later Credits for Reordering Refinements

- Expression e contextually refines expression e', denoted by $e \leq_{\text{ctx}} e'$, if no well-typed context C can distinguish the two.
- *Reordering refinement:* Execution order of e_1 and e_2 is not observable.

$$(e_1 \mid\mid e_2) \leq_{\text{ctx}} (e_1, e_2)$$

Later Credits for Reordering Refinements

- Expression e contextually refines expression e', denoted by $e \leq_{\text{ctx}} e'$, if no well-typed context C can distinguish the two.
- *Reordering refinement:* Execution order of e_1 and e_2 is not observable.

$$(e_1 \mid\mid e_2) \leq_{\text{ctx}} (e_1, e_2)$$

- ReLoC: Show contextual refinement using step-indexed logical relation
  - Right-hand program represented by *ghost state*

Later Credits for Reordering Refinements

- Expression e contextually refines expression e', denoted by $e \leq_{\text{ctx}} e'$, if no well-typed context C can distinguish the two.
- *Reordering refinement:* Execution order of e_1 and e_2 is not observable.

$$(e_1 \mid\mid e_2) \leq_{\text{ctx}} (e_1, e_2)$$

- ReLoC: Show contextual refinement using step-indexed logical relation
  - Right-hand program represented by *ghost state*
- In the paper: Reorderability of operations for JavaScript-style promises
  - Higher-order state!

Later Credits: Results

- Step-indexing and the later modality are powerful tools for modelling and reasoning about type systems and languages with "cyclic features"...

Later Credits: Results

- Step-indexing and the later modality are powerful tools for modelling and reasoning about type systems and languages with "cyclic features"...
- ...but they come at a price: The *proof burden* of challenging later eliminations.

Later Credits: Results

- Step-indexing and the later modality are powerful tools for modelling and reasoning about type systems and languages with "cyclic features"...
- ...but they come at a price: The *proof burden* of challenging later eliminations.

Later credits address this issue and facilitate:

- Simplifications of existing proofs
- New, previously unfeasible proofs

Acknowledgements

- Many thanks to Robbert for the guidance and support
- Thank you for your attention!

References I

[1] Simon Spies et al. "Later Credits: Resourceful Reasoning for the Later Modality". In: *Proc. ACM Program. Lang.* 6.ICFP (Aug. 2022). DOI: 10.1145/3547631. URL: https://doi.org/10.1145/3547631.

[2] Ralf Jung et al. "Iris from the ground up: A modular foundation for higher-order concurrent separation logic". In: *Journal of Functional Programming* 28 (2018), e20. DOI: 10.1017/S0956796818000151.

[3] Andrew W. Appel et al. "A very modal model of a modern, major, general type system". English (US). In: *Conference Record of POPL 2007.* Conference Record of the Annual ACM Symposium on Principles of Programming Languages. 2007, pp. 109–122. ISBN: 1595935754. DOI: 10.1145/1190216.1190235.

[4] Arthur Charguéraud. "Separation Logic for Sequential Programs (Functional Pearl)". In: 4.ICFP (Aug. 2020). DOI: 10.1145/3408998. URL: https://doi.org/10.1145/3408998.

[5] Derek Dreyer, Amal Ahmed and Lars Birkedal. "Logical Step-Indexed Logical Relations". In: *Logical Methods in Computer Science* Volume 7, Issue 2 (June 2011). DOI: 10.2168/LMCS-7(2:16)2011. URL: https://lmcs.episciences.org/698.

References II

[6] Kasper Svendsen, Filip Sieczkowski and Lars Birkedal. "Transfinite Step-Indexing: Decoupling Concrete and Logical Steps". In: *Programming Languages and Systems.* Ed. by Peter Thiemann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 727–751. ISBN: 978-3-662-49498-1.