# Deadlock-Free Separation Logic

## Linearity Yields Progress for Dependent Higher-Order Message Passing

Jules Jacobs (Radboud → Cornell)
Jonas Kastberg Hinrichsen (Aarhus)
Robbert Krebbers (Radboud)

# Our goal today:

Actris/Iris

**Separation Logic for Message Passing**
***with***
**Deadlock Freedom for Free**

Linear session types

## Desired adequacy theorem:

**For all e, if { Emp } e { Emp } can be derived, then e, when run, does not deadlock**

# Linear Session Types

**c : !int. ?bool. end**

## Deadlock freedom for message passing “for free” from type checking

**Caires, Pfenning, Carbone, Debois, Wadler, Gay, Vasconcelos, Lindley, Morris, etc.**

# Type Systems versus Program Logics

## Type systems

- Safety
- Automatic checking
- Ownership tied to values

## Program Logics

- Functional correctness
- Manual proof
- Ownership separate from values

# What is Iris?

Coq

## HeapLang {P} e {Q}

```
Definition newlock : val := λ: <>, ref #false.

Definition acquire : val := rec: "acquire" "l" :=
    if: CAS "l" #false #true then #() else "acquire" “l".

Definition release : val := λ: "l", "l" <- #false.
```

## iProp {P} e {Q}

```
Definition lock_inv (γ : gname) (l : loc) (R : iProp Σ) : iProp Σ :=
    ∃ b : bool, l ↦ #b ∗ if b then True else own γ (Excl ()) ∗ R.

Definition is_lock (γ : gname) (lk : val) (R : iProp Σ) : iProp Σ :=
    ∃ l: loc, ⌜lk = #l⌝ ∧ inv N (lock_inv γ l R).

Definition locked (γ : gname) : iProp Σ := own γ (Excl ()).
```

## Hoare Triples {P} e {Q}

```
Lemma acquire_spec γ lk R :
  {{{ is_lock γ lk R }}} acquire lk {{{ RET #(); locked γ ∗ R }}}.
Proof.
  iIntros (Φ) "#Hl HΦ". iLöb as "IH". wp_rec.
  wp_apply (try_acquire_spec with "Hl"). iIntros ([]).
  - iIntros "[Hlked HR]". wp_if. iApply "HΦ"; auto with iFrame.
  - iIntros "_". wp_if. iApply ("IH" with "[HΦ]"). auto.
Qed.
```

## Iris Proof Mode

# Actris: message passing in Iris

Jonas Kastberg Hinrichsen, Jesper Bengtson, Robbert Krebbers

(POPL'20, LMCS'22), et al. (CPP'21, 2x ICFP'23)

```
{ Emp }
 c,d := new_chan()
 fork { c.send(c.recv() + 1) }
 d.send(2)
 assert(d.recv() == 3)
{ Emp }
```

```
c ↣ ?(n : nat)<n>. !<n+1>. end

d ↣ !(n : nat)<n>. ?<n+1>. end
```

```
{ Emp }
 c,d := new_chan(); r = ref(0)
 fork { r += c.recv(); c.send(1) }
 d.send(2); r += d.recv()
 assert(!r == 3)
{ Emp }
```

```
c ↣ ?(n m : nat)<n>{ r ↦ m }.
      !<1>{ r ↦ m+n }. end

d ↣ !(n m : nat)<n>{ r ↦ m }.
      ?<1>{ r ↦ m+n }. end
```

# Problem

## Actris is not sensitive to deadlocks

```
{ Emp }
 c,d := new_chan(); r = ref(0)
 fork { r += c.recv(); c.send(1) }
 d.send(2); r += d.recv()
 assert(!r == 3)
{ Emp }
```

```
{ Emp }
 c1,d1 := new_chan()
 c2,d2 := new_chan()
 fork { c1.recv(); d2.send(2) }
 c2.recv(); d1.send(3)
{ Emp }
```

**Verification goes through, even for deadlocks!**

**So what does { Emp } e { Emp } mean?**

# Iris' adequacy theorem

{ Emp } e { Emp } → no thread gets stuck

("safety")

**Partial correctness: e can loop**

**In the depths of Actris' recv operation, we find…**

```
rec: "acquire" "l" :=
  if: CAS "l" #false #true then #() else "acquire" "l".
```

**Deadlock not distinguished from busy spin loop**

# Contribution

## LinearActris

### A separation logic for deadlock-free message passing

```
{ Emp }
 c,d := new_chan(); r = ref(0)
 r += c.recv(); c.send(1)
 d.send(2); r += d.recv()
 assert(!r == 3)
{ Emp }
```

✗

```
{ Emp }
 c,d := new_chan(); r = ref(0)
 fork { r += c.recv(); c.send(1) }
 d.send(2); r += d.recv()
 assert(!r == 3)
{ Emp }
```

✓

# Deadlock sensitive operational semantics

**Our c.recv() is <u>primitive</u> and gets <u>stuck</u> until a message arrives**

**Desired adequacy theorems:**

**{ Emp } e { Emp } → no thread gets stuck, except by c.recv()**

("safety")

**{ Emp } e { Emp } → configuration as a whole never gets stuck**

("global progress")

**(Bonus: no leaked memory 😌)**

# Changes to the Iris/Actris proof rules

**Change 1:** make the logic *linear*

Cannot drop obligation d ↣ !(n : nat)<n>…

**Change 2:** combine new_chan with fork

Use c = fork_chan { d => … } like session types

**Change 3:** remove Iris invariants 😭

## → LinearActris is now deadlock free!

# Deadlock attempt

```
c,d := new_chan(); r = ref(0)
fork { r += c.recv(); c.send(1) }
d.send(2); r += d.recv()
assert(!r == 3)
```

Not possible with fork_chan!

Let's try to be clever:

```
r = ref(0)
c = fork_chan { d => r := d }
d = !r; r := 0
r += c.recv(); c.send(1)
d.send(2); r += d.recv()
assert(!r == 3)
```

```
r := ref(0)
c = fork_chan { d => d.send(d) }
d = c.recv()
r += c.recv(); c.send(1)
d.send(2); r += d.recv()
assert(!r == 3)
```

Escape attempts do not work!

# Key challenge: **proving that escape is futile**

**Channels *are* first-class values**

```
{ Emp }
  r = ref(0)
  c1 = fork_chan { d1 => n = d1.recv(); !r.send(n+1) }
  c2 = fork_chan { d2 => assert(d2.recv() == 3) }
  r := c2
  c1.send(2)
{ Emp }
```

✓

# Adequacy proof

- Iris: one shared resource world for all threads
- LinearActris: separate resource world per thread & channel, connected by ownership graph

| Thread 1 | Thread 1 | Thread 2 |
| --- | --- | --- |
| (c ↣ !(n:nat)…) ∗ (c ↣ ?(n:nat)…) ⊢ False | (c ↣ !(n:nat)…) | (c ↣ ?(n:nat)…) ✓ |

- Invariant: ownership graph acyclic
- Higher-order & infinite protocols via step-indexing

c ↣ !(n:nat)<n>{ d ↣ ?(b:bool)<b>… }…

→ see paper

# LinearActris architecture

Coq → **LinearActris**

## ChanLang {P} e {Q}

+ Functional programming
+ Mutable references
+ Message passing concurrency

## Actris-style session protocols

+ Stateful dependent protocols
+ Send resources & channels
+ Infinite protocols
+ …

## aProp {P} e {Q}

+ Linear separation logic
+ Heap ownership
+ Channel ownership
+ No Iris invariants 😢

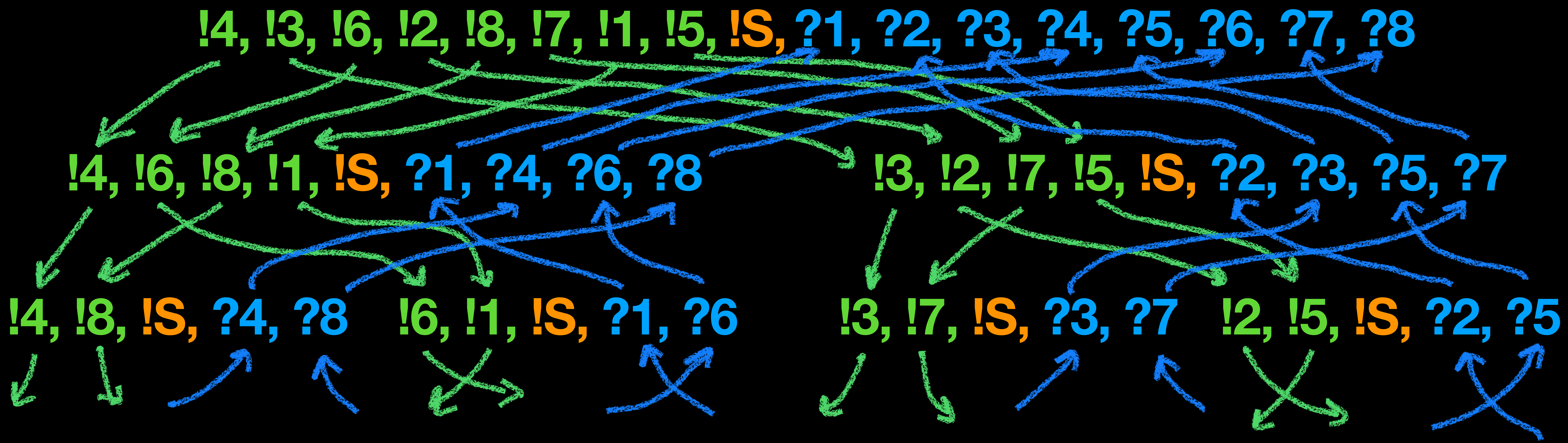
## Hoare Triples {P} e {Q}

```
Lemma prog_spec c : {{{ c ↣ prot }}} prog #() {{{ RET #(); emp }}}.
Proof.
  iIntros (Φ) "_ HΦ".
  wp_send. wp_recv. iApply wp_assert. wp_pures. iSplit; [done|].
  iIntros "!>". wp_wait. by iApply "HΦ".
Qed.
```

## Iris Proof Mode

**+ safety**
**+ global progress**

# Verification Example: Actris Merge Sort




A big concurrent mess…

…but deadlock freedom comes for free

*No additional proof obligations!*

# Can we verify every session-typed program?

# Embedding session types: semantic typing

## Step 1: a session type system for ChanLang

(+ mutable references, polymorphism, recursion, etc.)

## Step 2: interpret types as LinearActris predicates

$[\![ T ]\!] : Val \to aProp$

## Step 3: prove fundamental lemma

$\vdash e : T \to \{ Emp \}\ e\ \{ [\![ T ]\!] \}$

## Step 4: apply adequacy

$\vdash e : T \to$ e is deadlock free

*Trivial proofs…*

*…but state of the art type system!*

# Deadlock-Free Separation Logic

## Linearity Yields Progress for Dependent Higher-Order Message Passing

**Jules Jacobs (Radboud → Cornell)**
**Jonas Kastberg Hinrichsen (Aarhus)**
**Robbert Krebbers (Radboud)**

# Questions?

# Future work:

- **Other primitives (e.g., locks)**
- **Even better: Iris invariants**
- **Liveness**