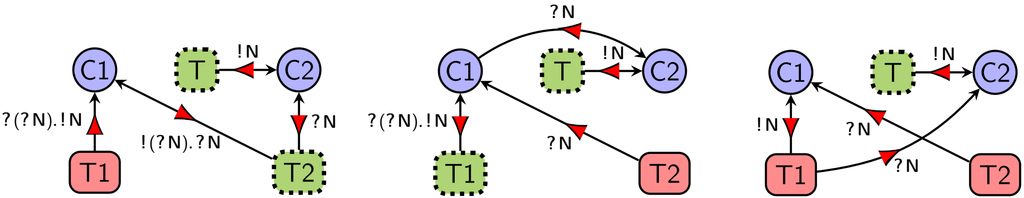
Mechanized Deadlock Freedom for Session Types

Jules Jacobs[1] Robbert Krebbers[1] Stephanie Balzer[2]



[1]Radboud University, The Netherlands
[2]Carnegie Mellon University, USA

VEST'21

Mechanization of binary session types

- State of the art: type safety
- Our goal: deadlock and leak freedom

Mechanization of binary session types

- State of the art: type safety
- Our goal: deadlock and leak freedom
- **Cyclic waiting dependency $\implies$ deadlock**
- **Reasoning about waiting dependencies in a proof assistant is hard**

Mechanization of binary session types

- State of the art: type safety
- Our goal: deadlock and leak freedom
- **Cyclic waiting dependency $\implies$ deadlock**
- **Reasoning about waiting dependencies in a proof assistant is hard**
- Our approach: connectivity graphs
  - Keeps track of type environments and reference topology simultaneously

Mechanization of binary session types

- State of the art: type safety
- Our goal: deadlock and leak freedom
- **Cyclic waiting dependency $\implies$ deadlock**
- **Reasoning about waiting dependencies in a proof assistant is hard**
- Our approach: connectivity graphs
  - Keeps track of type environments and reference topology simultaneously
- **Develop tools for $\mathit{Cgraph}(V, L)$ abstract in nodes V and edge labels L, that encapsulate the graph reasoning:**
  1. **Well-formedness:** link the Cgraph with the configuration using separation logic
  2. **Progress:** waiting induction principle
  3. **Preservation:** separation logic local graph transformations

Thread pool:

```
⇒let c1 : !(?N.)?N. =
   fork (λ  c1’,
     let (c1’,c) = recv(c1’)
     let (c,n) = recv(c);
     let c1’ = send(c1’,n)
     close(c); close(c1’))

 let c2 : !N. =
   fork (λ  c2’,
     let c1 = send(c1,c2’);
     let (c1,m) = recv(c1))
     close(c1))

 let c2 = send(c2,10);
 close(c2)
```

Heap:

Connectivity graph:

Thread pool:

```
⇒let c1 = #1_L

 let c2 : !N. =
   fork (λ  c2',
     let c1 = send(c1,c2');
     let (c1,m) = recv(c1)
     close(c1))

 let c2 = send(c2,10);
 close(c2)
```

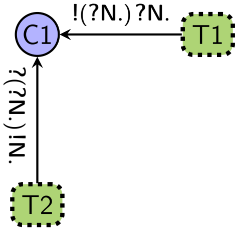
```
 (λ  c1',
   let (c1',c) = recv(c1')
   let (c,n) = recv(c);
   let c1' = send(c1',n)
   close(c); close(c1')) #1_R
```

Heap:

$\#1_L \mapsto [\,]$

$\#1_R \mapsto [\,]$

Connectivity graph:



Thread pool:

```
let c2 : !N. =
  fork (λ  c2',
    let c1 = send(#1_L,c2');
    let (c1,m) = recv(c1)
    close(c1))

let c2 = send(c2,10);
close(c2)
```

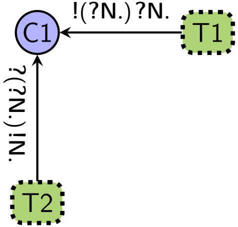
```
⇒(λ  c1',
  let (c1',c) = recv(c1')
  let (c,n) = recv(c);
  let c1' = send(c1',n)
  close(c); close(c1')) #1_R
```

Heap:

$\#1_L \mapsto [\,]$

$\#1_R \mapsto [\,]$

Connectivity graph:



Thread pool:

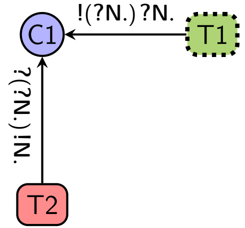
```
⇒let c2 : !N. =
   fork (λ  c2',
     let c1 = send(#1_L,c2');
     let (c1,m) = recv(c1)
     close(c1))

  let c2 = send(c2,10);
  close(c2)
  ______________________

▶ let (c1',c) = recv(#1_R)
  let (c,n) = recv(c);
  let c1' = send(c1',n)
  close(c); close(c1')
```

Heap:

$\#1_L \mapsto [\,]$

$\#1_R \mapsto [\,]$

Connectivity graph:



Thread pool:

`let c2 =` $\#2_L$

`let c2 = send(c2,10);`
`close(c2)`

▶ `let (c1’,c) = recv(`$\#1_R$`)`
`let (c,n) = recv(c);`
`let c1’ = send(c1’,n)`
`close(c); close(c1’)`

$\Rightarrow$ `(`λ `c2’,`
`let c1 = send(`$\#1_L$`,c2’);`
`let (c1,m) = recv(c1)`
`close(c1))` $\#2_R$

Heap:

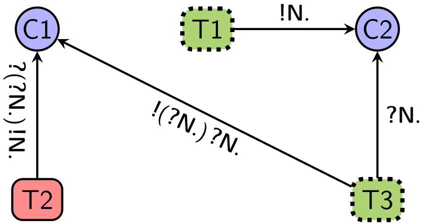
$\#1_L \mapsto [\,]$

$\#1_R \mapsto [\,]$

$\#2_L \mapsto [\,]$

$\#2_R \mapsto [\,]$

Connectivity graph:



Thread pool:

```
⇒let c2 = #2_L

 let c2 = send(c2,10);
 close(c2)
```

```
► let (c1',c) = recv(#1_R)
  let (c,n) = recv(c);
  let c1' = send(c1',n)
  close(c); close(c1')
```

```
  let c1 = send(#1_L,#2_R);
  let (c1,m) = recv(c1)
  close(c1)
```

Heap:

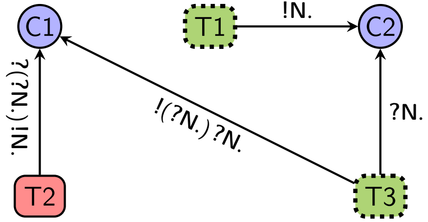
$\#1_L \mapsto [\,]$

$\#1_R \mapsto [\,]$

$\#2_L \mapsto [\,]$

$\#2_R \mapsto [\,]$

Connectivity graph:



Thread pool:

```
let c2 = send(#2_L,10);
close(c2)
```

```
▶ let (c1',c) = recv(#1_R)
let (c,n) = recv(c);
let c1' = send(c1',n)
close(c); close(c1')
```

```
⇒ let c1 = send(#1_L,#2_R);
let (c1,m) = recv(c1)
close(c1)
```

Heap:

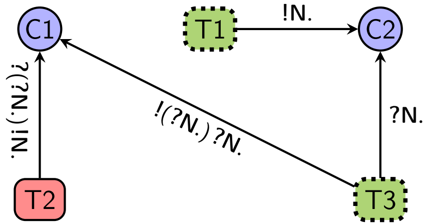
$\#1_L \mapsto [\,]$

$\#1_R \mapsto [\,]$

$\#2_L \mapsto [\,]$

$\#2_R \mapsto [\,]$

Connectivity graph:



Thread pool:

```
let c2 = send(#2_L,10);
close(c2)
```

```
⇒let (c1’,c) = recv(#1_R)
 let (c,n) = recv(c);
 let c1’ = send(c1’,n)
 close(c); close(c1’)
```

```
▶ let (c1,m) = recv(#1_L)
  close(c1)
```

Heap:

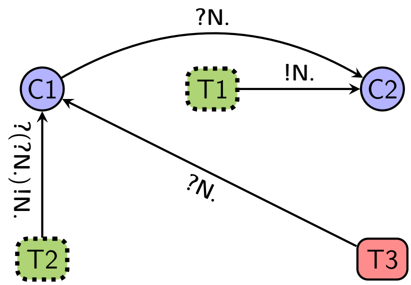
$\#1_L \mapsto [\,]$

$\#1_R \mapsto [\#2_R]$

$\#2_L \mapsto [\,]$

$\#2_R \mapsto [\,]$

Connectivity graph:



Thread pool:

```
let c2 = send(#2_L,10);
close(c2)
```

```
⇒let (c1',c) = (#1_R,#2_R)
 let (c,n) = recv(c);
 let c1' = send(c1',n)
 close(c); close(c1')
```

```
▶ let (c1,m) = recv(#1_L)
  close(c1)
```

Heap:

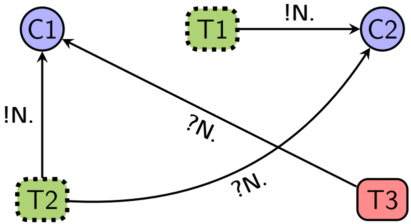
$\#1_L \mapsto [\,]$

$\#1_R \mapsto [\,]$

$\#2_L \mapsto [\,]$

$\#2_R \mapsto [\,]$

Connectivity graph:



Thread pool:

```
⇒let c2 = send(#2_L,10);
  close(c2)
```

```
▶ let (c,n) = recv(#2_R);
  let c1' = send(#1_R,n)
  close(c); close(c1')
```

```
▶ let (c1,m) = recv(#1_L)
  close(c1)
```

Heap:

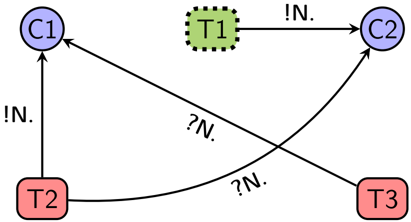
$\#1_L \mapsto [\,]$

$\#1_R \mapsto [\,]$

$\#2_L \mapsto [\,]$

$\#2_R \mapsto [\,]$

Connectivity graph:



Thread pool:

⇒ close(#2_L)

```
let (c,n) = recv(#2_R);
let c1' = send(#1_R,n)
close(c); close(c1')
```

```
► let (c1,m) = recv(#1_L)
close(c1)
```

Heap:

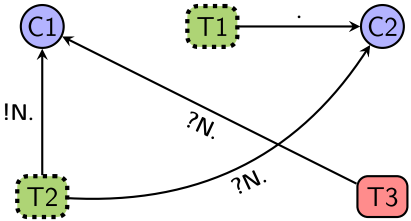
$\#1_L \mapsto [\,]$

$\#1_R \mapsto [\,]$

$\#2_L \mapsto [\,]$

$\#2_R \mapsto [10]$

Connectivity graph:



Thread pool:

```
()
```

```
⇒ let (c,n) = recv(#2_R);
  let c1' = send(#1_R,n)
  close(c); close(c1')
```

```
▶ let (c1,m) = recv(#1_L)
  close(c1)
```

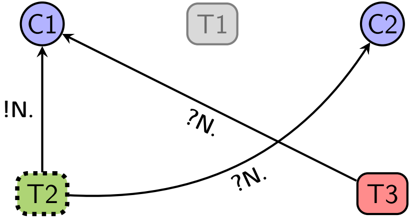
Heap:

$\#1_L \mapsto [\,]$

$\#1_R \mapsto [\,]$

$\#2_R \mapsto [10]$

Connectivity graph:



Thread pool:

```
()
```

```
⇒let c1' = send(#1_R,10)
 close(#2_R); close(c1')
```

```
▶ let (c1,m) = recv(#1_L)
  close(c1)
```

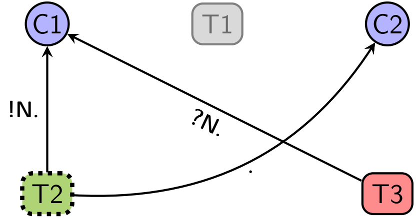
Heap:

$\#1_L \mapsto [\,]$

$\#1_R \mapsto [\,]$

$\#2_R \mapsto [\,]$

Connectivity graph:



Thread pool:

```
()
```

⇒ `close(#2`$_R$`); close(#1`$_R$`)`

`let (c1,m) = recv(#1`$_L$`)`
`close(c1)`

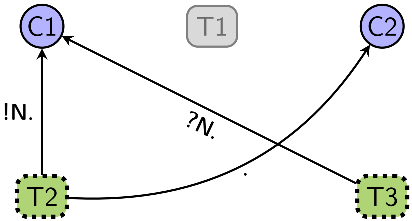
Heap:

$\#1_L \mapsto [10]$

$\#1_R \mapsto [\ ]$

$\#2_R \mapsto [\ ]$

Connectivity graph:



Thread pool:

```
()
```

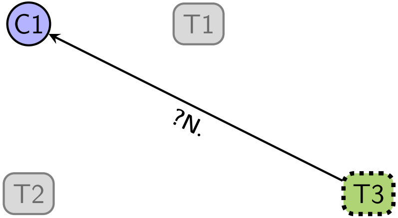
```
()
```

⇒ `let (c1,m) = recv(`$\#1_L$`)`
`close(c1)`

Heap:

$\#1_L \mapsto [10]$

Connectivity graph:



Thread pool:

()

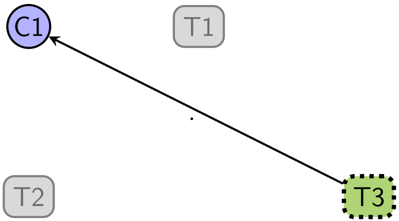
()

$\Rightarrow$ `close`($\#1_L$)

Heap:

$\#1_L \mapsto [\,]$

Connectivity graph:



Thread pool:

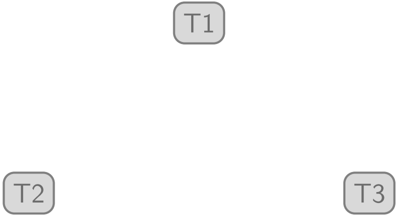
()

()

()

Heap:

Connectivity graph:



Thread pool:

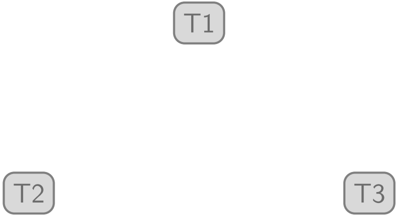
()

()

()

Heap:

Connectivity graph:



Theorem (Deadlock and memory leak freedom)
If the configuration cannot step, then all threads are () and the heap is empty.

Thread pool:

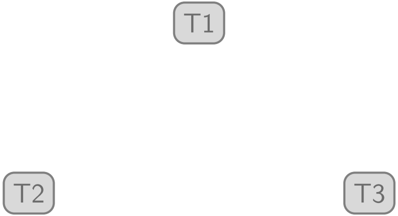
()

()

()

Heap:

Connectivity graph:



Theorem (Deadlock and memory leak freedom)

If the configuration cannot step, then all threads are () and the heap is empty.

Theorem (Progress)

If the configuration is well-formed, and if there exists a non-() thread or a buffer in the heap, then the configuration can step.

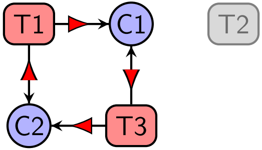
Theorem (Preservation)

Well-formed configurations step to well-formed configurations.

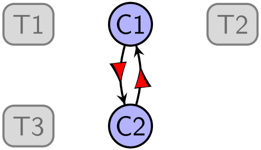
```
// no counter party
let c1 = fork(λ c1’, ())
receive(c1)
```


```
// protocol violation
let c1 = fork(λ c1’, receive(c1’); ...)
receive(c1)
```

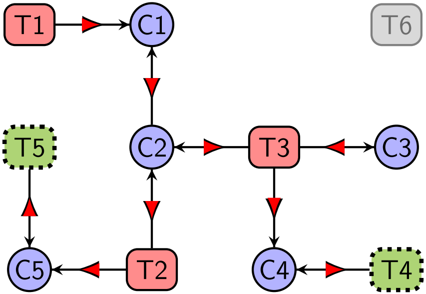

```
// circular dependency
let c1 = fork(λ c1’, send(c1’,c1’))
let (c1,c1’) = receive(c1)
let c2 = fork(λ c2’,
  let (c2’,v) = receive(c2’)
  send(c1’,v); ...)
let (c1,v) = receive(c1) in send(c2,v)
```



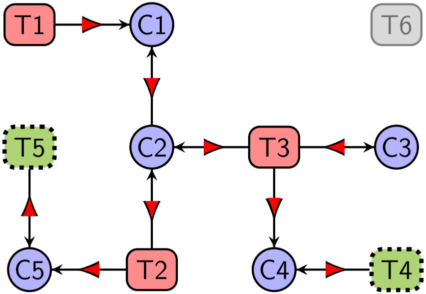

```
// memory leak
let c1 = fork(λ c1’, ()))
let c2 = fork(λ c2’, ())
send(c2,c1)
send(c1,c2)
```



Waiting induction principle



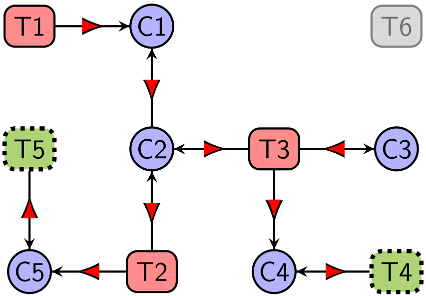
Waiting induction principle



Lemma (Waiting induction)

Assume that the undirected erasure of the graph is acyclic.
To prove $P(v)$, we may assume that $v \blacktriangleright w \implies P(w)$ for all $w \in V$.

Waiting induction principle

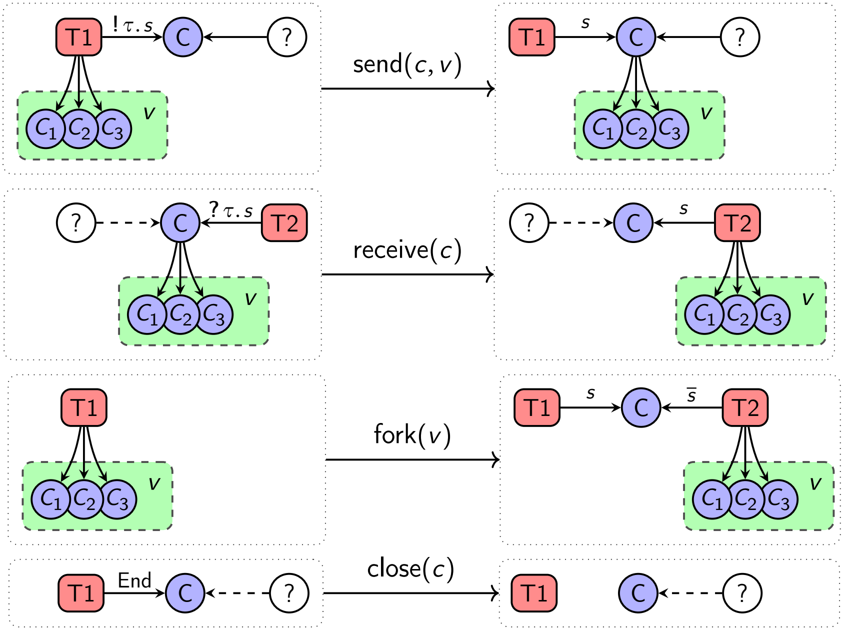


Lemma (Waiting induction)

Assume that the undirected erasure of the graph is acyclic.
To prove $P(v)$, we may assume that $v \blacktriangleright w \implies P(w)$ for all $w \in V$.

Graph acyclicity reasoning is encapsulated in *generic* waiting induction.

- The progress proof does local, language specific reasoning.


T1
!τ.s
C
?
C_1 C_2 C_3
v
send(c, v)
s
receive(c)
?τ.s
T2
fork(v)
$\bar{s}$
End
close(c)

$$\frac{\Gamma = \{x \mapsto \tau\}}{\Gamma \vdash x : \tau} \qquad \frac{.}{\emptyset \vdash () : \mathbf{1}} \qquad \frac{n \in \mathbb{N}}{\emptyset \vdash n : \mathbf{N}} \qquad \frac{\Gamma_1 \vdash e_1 : \tau_1 \qquad \Gamma_2 \vdash e_2 : \tau_2}{\Gamma_1 \uplus \Gamma_2 \vdash (e_1, e_2) : \tau_1 \times \tau_2}$$

$$\frac{\Gamma \uplus \{x \mapsto \tau_1\} \vdash e : \tau_2}{\Gamma \vdash \lambda x.\ e : \tau_1 \multimap \tau_2} \qquad \frac{\Gamma_1 \vdash e_1 : \tau_1 \multimap \tau_2 \qquad \Gamma_2 \vdash e_2 : \tau_1}{\Gamma_1 \uplus \Gamma_2 \vdash e_1\ \ e_2 : \tau_2}$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \qquad \Gamma_2 \uplus \{x \mapsto \tau_1\} \vdash e_2 : \tau_2}{\Gamma_1 \uplus \Gamma_2 \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau_2} \qquad \frac{\Gamma_1 \vdash e_1 : \mathbf{N} \qquad \Gamma_2 \vdash e_2 : \tau \qquad \Gamma_2 \vdash e_3 : \tau}{\Gamma_1 \uplus \Gamma_2 \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau}$$

$$\frac{\Gamma \vdash e : \overline{s} \multimap \mathbf{1}}{\Gamma \vdash \mathsf{fork}(e) : s} \qquad \frac{\Gamma_1 \vdash e_1 : \mathbf{!}\,\tau.\,s \qquad \Gamma_2 \vdash e_2 : \tau}{\Gamma_1 \uplus \Gamma_2 \vdash \mathsf{send}(e_1, e_2) : s} \qquad \frac{\Gamma \vdash e : \mathbf{?}\,\tau.\,s}{\Gamma \vdash \mathsf{receive}(e) : s \times \tau} \qquad \frac{\Gamma \vdash e : \mathsf{End}}{\Gamma \vdash \mathsf{close}(e) : \mathbf{1}}$$

$$\frac{\ulcorner \Gamma = \{x \mapsto \tau\} \urcorner}{\Gamma \vdash x : \tau}\,*$$

$$\frac{\mathsf{Emp}}{\emptyset \vdash () : \mathbf{1}}\,*$$

$$\frac{\ulcorner n \in \mathbb{N} \urcorner}{\emptyset \vdash n : \mathbf{N}}\,*$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \quad * \quad \Gamma_2 \vdash e_2 : \tau_2}{\Gamma_1 \uplus \Gamma_2 \vdash (e_1, e_2) : \tau_1 \times \tau_2}\,*$$

$$\frac{\Gamma \uplus \{x \mapsto \tau_1\} \vdash e : \tau_2}{\Gamma \vdash \lambda x.\ e : \tau_1 \multimap \tau_2}\,*$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \multimap \tau_2 \quad * \quad \Gamma_2 \vdash e_2 : \tau_1}{\Gamma_1 \uplus \Gamma_2 \vdash e_1\ e_2 : \tau_2}\,*$$

$$\frac{\Gamma_1 \vdash e_1 : \tau_1 \quad * \quad \Gamma_2 \uplus \{x \mapsto \tau_1\} \vdash e_2 : \tau_2}{\Gamma_1 \uplus \Gamma_2 \vdash \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 : \tau_2}\,*$$

$$\frac{\Gamma_1 \vdash e_1 : \mathbf{N} \quad * \quad (\Gamma_2 \vdash e_2 : \tau \quad \wedge \quad \Gamma_2 \vdash e_3 : \tau)}{\Gamma_1 \uplus \Gamma_2 \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 : \tau}\,*$$

$$\frac{\Gamma \vdash e : \bar{s} \multimap 1}{\Gamma \vdash \mathsf{fork}(e) : s}\,*$$

$$\frac{\Gamma_1 \vdash e_1 : \,!\,\tau.s \quad * \quad \Gamma_2 \vdash e_2 : \tau}{\Gamma_1 \uplus \Gamma_2 \vdash \mathsf{send}(e_1, e_2) : s}\,*$$

$$\frac{\Gamma \vdash e : \,?\,\tau.s}{\Gamma \vdash \mathsf{receive}(e) : s \times \tau}\,*$$

$$\frac{\Gamma \vdash e : \mathsf{End}}{\Gamma \vdash \mathsf{close}(e) : \mathbf{1}}\,*$$

$$\frac{\mathsf{own}(\mathsf{Chan}(a) \mapsto (t, s))}{\emptyset \vdash \#a_t : s}\,*$$

$$\left(\frac{\text{Arjen Rouvoet}}{\Gamma \vdash e : \tau : iProp}\,*\right)$$

Connectivity graphs:

$$Cgraph(V, L) \triangleq \{G \in V \times V \xrightarrow{\text{fin}} L \mid G \text{ has no undirected cycles}\}$$

Connectivity graphs:

$$Cgraph(V, L) \triangleq \{G \in V \times V \xrightarrow{\text{fin}} L \mid G \text{ has no undirected cycles}\}$$

Instantiation:

$$v \in V ::= \mathrm{Thread}(i) \mid \mathrm{Chan}(a) \qquad\qquad l \in L \triangleq \{0, 1\} \times Session$$

Connectivity graphs:

$$Cgraph(V, L) \triangleq \{G \in V \times V \xrightarrow{\text{fin}} L \mid G \text{ has no undirected cycles}\}$$

Instantiation:

$$v \in V ::= \text{Thread}(i) \mid \text{Chan}(a) \qquad\qquad l \in L \triangleq \{0, 1\} \times Session$$

Generic well-formedness:

$$\text{wf}(P) \triangleq \exists G : Cgraph(V, L).\ \forall v \in V.\ P(v, \text{in}(G, v))(\text{out}(G, v))$$

Connectivity graphs:

$$Cgraph(V, L) \triangleq \{G \in V \times V \xrightarrow{\text{fin}} L \mid G \text{ has no undirected cycles}\}$$

Instantiation:

$$v \in V ::= \text{Thread}(i) \mid \text{Chan}(a) \qquad\qquad l \in L \triangleq \{0, 1\} \times Session$$

Generic well-formedness:

$$\text{wf}(P) \triangleq \exists G : Cgraph(V, L).\ \forall v \in V.\ P(v, \text{in}(G, v))(\text{out}(G, v))$$

Local well-formedness predicate:

$$P : V \times Multiset\ L \to (V \xrightarrow{\text{fin}} L) \to Prop$$

P can talk about **incoming label multiset** and **outgoing edges**.

Connectivity graphs:

$$Cgraph(V, L) \triangleq \{G \in V \times V \xrightarrow{\text{fin}} L \mid G \text{ has no undirected cycles}\}$$

Instantiation:

$$v \in V ::= \mathrm{Thread}(i) \mid \mathrm{Chan}(a) \qquad l \in L \triangleq \{0, 1\} \times Session$$

Generic well-formedness:

$$\mathrm{wf}(P) \triangleq \exists G : Cgraph(V, L).\, \forall v \in V.\, P(v, \mathrm{in}(G, v))(\mathrm{out}(G, v))$$

Local well-formedness predicate:

$$P : V \times Multiset\, L \to (V \xrightarrow{\text{fin}} L) \to Prop$$

P can talk about **incoming label multiset** and **outgoing edges**.

- Threads: expression is well-typed w.r.t. the session types on its outgoing edges.

Connectivity graphs:

$$Cgraph(V, L) \triangleq \{G \in V \times V \xrightarrow{\text{fin}} L \mid G \text{ has no undirected cycles}\}$$

Instantiation:

$$v \in V ::= \text{Thread}(i) \mid \text{Chan}(a) \qquad\qquad l \in L \triangleq \{0, 1\} \times Session$$

Generic well-formedness:

$$\text{wf}(P) \triangleq \exists G : Cgraph(V, L).\ \forall v \in V.\ P(v, \text{in}(G, v))(\text{out}(G, v))$$

Local well-formedness predicate:

$$P : V \times Multiset\ L \to (V \xrightarrow{\text{fin}} L) \to Prop$$

P can talk about **incoming label multiset** and **outgoing edges**.

- Threads: expression is well-typed w.r.t. the session types on its outgoing edges.
- Channels: the two incoming labels are dual up to the values in the buffers.

Connectivity graphs:

$$Cgraph(V, L) \triangleq \{G \in V \times V \xrightarrow{\text{fin}} L \mid G \text{ has no undirected cycles}\}$$

Instantiation:

$$v \in V ::= \mathrm{Thread}(i) \mid \mathrm{Chan}(a) \qquad\qquad l \in L \triangleq \{0, 1\} \times Session$$

Generic well-formedness:

$$\mathrm{wf}(P) \triangleq \exists G : Cgraph(V, L).\ \forall v \in V.\ P(v, \mathrm{in}(G, v))(\mathrm{out}(G, v))$$

Local well-formedness predicate:

$$P : V \times Multiset\ L \rightarrow (V \xrightarrow{\text{fin}} L) \rightarrow Prop$$

P can talk about **incoming label multiset** and **outgoing edges.**

- Threads: expression is well-typed w.r.t. the session types on its outgoing edges.
- Channels: the two incoming labels are dual up to the values in the buffers.
- Intuition: outgoing edges = who we own, incoming edges = who owns us, **and at which types.**

Connectivity graphs:

$$Cgraph(V, L) \triangleq \{G \in V \times V \xrightarrow{\text{fin}} L \mid G \text{ has no undirected cycles}\}$$

Instantiation:

$$v \in V ::= \mathrm{Thread}(i) \mid \mathrm{Chan}(a) \qquad\qquad l \in L \triangleq \{0, 1\} \times Session$$

Generic well-formedness:

$$\mathrm{wf}(P) \triangleq \exists G : Cgraph(V, L).\ \forall v \in V.\ P(v, \mathrm{in}(G, v))(\mathrm{out}(G, v))$$

Local well-formedness predicate:

$$P : V \times Multiset\ L \to (V \xrightarrow{\text{fin}} L) \to Prop$$

P can talk about **incoming label multiset** and **outgoing edges.**

- Threads: expression is well-typed w.r.t. the session types on its outgoing edges.
- Channels: the two incoming labels are dual up to the values in the buffers.
- Intuition: outgoing edges = who we own, incoming edges = who owns us, **and at which types.**
- P connects $\mathrm{out}(G, v)$ and $\mathrm{in}(G, v)$ with the local configuration state of v.

Connectivity graphs:

$$\mathit{Cgraph}(V, L) \triangleq \{G \in V \times V \xrightarrow{\text{fin}} L \mid G \text{ has no undirected cycles}\}$$

Instantiation:

$$v \in V ::= \mathsf{Thread}(i) \mid \mathsf{Chan}(a) \qquad\qquad l \in L \triangleq \{0, 1\} \times \mathit{Session}$$

Generic well-formedness:

$$\mathsf{wf}(P) \triangleq \exists G : \mathit{Cgraph}(V, L).\ \forall v \in V.\ P(v, \mathsf{in}(G, v))(\mathsf{out}(G, v))$$

Local well-formedness predicate:

$$P : V \times \mathit{Multiset}\ L \to \mathit{iProp}$$

P can talk about **incoming label multiset** and **outgoing edges.**

- Threads: expression is well-typed w.r.t. the session types on its outgoing edges.
- Channels: the two incoming labels are dual up to the values in the buffers.
- Intuition: outgoing edges = who we own, incoming edges = who owns us,
  and at which types.
- P connects $\mathsf{out}(G, v)$ and $\mathsf{in}(G, v)$ with the local configuration state of v.
- Separation logic: $\mathit{iProp} \triangleq (V \xrightarrow{\text{fin}} L) \to \mathit{Prop}$

Graph transformations in separation logic

Lemmas for maintaining $\mathsf{wf}(P)$ when adding, removing, and relabeling edges, and **exchanging** separation logic resources.

Graph transformations in separation logic

Lemmas for maintaining $\mathrm{wf}(P)$ when adding, removing, and relabeling edges, and **exchanging** separation logic resources.

Lemma (Exchange)

Let $v_1, v_2 \in V$. To prove $\mathrm{wf}(P)$ *implies* $\mathrm{wf}(P')$*, it suffices to prove:*

1. $P(v, \Delta) \mathrel{-\!\!*} P'(v, \Delta)$ *for all* $v \in V \setminus \{v_1, v_2\}$ *and* $\Delta \in \mathit{Multiset}\ L$
2. $P(v_1, \Delta_1) \mathrel{-\!\!*} \exists l.\, \mathrm{own}(v_2 \mapsto l) * \forall \Delta_2 \in \mathit{Multiset}\ L.\ P(v_2, \{l\} \uplus \Delta_2)$
   $\qquad \mathrel{-\!\!*} \exists l'.\, (\mathrm{own}(v_2 \mapsto l') \mathrel{-\!\!*} P'(v_1, \Delta_1)) * P'(v_2, \{l'\} \uplus \Delta_2)$
   for all $\Delta_1 \in \mathit{Multiset}\ L$

Preservation proof appears to do no graph reasoning at all!

- The construction of the new connectivity graph, and the proof of its acyclicity, is encapsulated in the *generic* lemmas.
- The preservation proof does only local, language specific reasoning about P.

Mechanization

Our language:

1. Functional (sums, products, closures, etc.) + session-typed channels
2. Linear and unrestricted types
   - Unrestricted: numbers, sums, products, unrestricted function type ($\rightarrow$)
   - Linear: channels, sums, products, linear function type ($\multimap$)
3. General recursive types: coinductive method adapted from Gay et al. [2020]
   - Recursive session types, including through the message
   - Algebraic data types using recursion + sums + products

Mechanization

Our language:

1. Functional (sums, products, closures, etc.) + session-typed channels
2. Linear and unrestricted types
   - Unrestricted: numbers, sums, products, unrestricted function type ($\rightarrow$)
   - Linear: channels, sums, products, linear function type ($\multimap$)
3. General recursive types: coinductive method adapted from Gay et al. [2020]
   - Recursive session types, including through the message
   - Algebraic data types using recursion + sums + products

Mechanization:

- $Cgraph(V, L)$ library: 4988 LOC
- Language definition: 447 LOC
- Deadlock and leak freedom proof: 2542 LOC

Mechanization

Our language:

1. Functional (sums, products, closures, etc.) + session-typed channels
2. Linear and unrestricted types
   - Unrestricted: numbers, sums, products, unrestricted function type ($\rightarrow$)
   - Linear: channels, sums, products, linear function type ($\multimap$)
3. General recursive types: coinductive method adapted from Gay et al. [2020]
   - Recursive session types, including through the message
   - Algebraic data types using recursion + sums + products

Mechanization:

- $\mathit{Cgraph}(V, L)$ library: 4988 LOC
- Language definition: 447 LOC
- Deadlock and leak freedom proof: 2542 LOC

Initial direct attempt: proofs goals got too complex.
Graph reasoning intertwined with language specifics.
Encapsulating the graph reasoning made it manageable.

Questions?

julesjacobs@gmail.com

These slides: julesjacobs.com/slides/vest2021.pdf