

Mechanised Functional Verification of Mixed Choice Multiparty Message Passing

Jonas Kastberg Hinrichsen

Aalborg University

Iwan Quémerais

ENS-Lyon

Lars Birkedal

Aarhus University

Accepted at OOPSLA'26 (Camera Ready Deadline 27. February)

10. February, Dagstuhl Seminar 26071

Leader Election

Given a set of actors, leader election is an algorithm that satisfies:

- ▶ **Uniqueness:** There is exactly one actor that considers itself as leader
- ▶ **Agreement:** All other actors know who the leader is
- ▶ **Termination:** The algorithm finishes in finite time

Leader Election

Given a set of actors, leader election is an algorithm that satisfies:

- ▶ **Uniqueness:** There is exactly one actor that considers itself as leader
- ▶ **Agreement:** All other actors know who the leader is
- ▶ **Termination:** The algorithm finishes in finite time

We lift the properties to functional correctness as:

- ▶ **Uniqueness:** The leader can proceed with elevated permissions
- ▶ **Agreement:** Participant interaction can depend on known leader

Leader Election

Given a set of actors, leader election is an algorithm that satisfies:

- ▶ **Uniqueness:** There is exactly one actor that considers itself as leader
- ▶ **Agreement:** All other actors know who the leader is
- ▶ **Termination:** The algorithm finishes in finite time

We lift the properties to functional correctness as:

- ▶ **Uniqueness:** The leader can proceed with elevated permissions
- ▶ **Agreement:** Participant interaction can depend on known leader

We consider process-level implementation details

Leader Election

Given a set of actors, leader election is an algorithm that satisfies:

- ▶ **Uniqueness:** There is exactly one actor that considers itself as leader
- ▶ **Agreement:** All other actors know who the leader is
- ▶ **Termination:** The algorithm finishes in finite time

We lift the properties to functional correctness as:

- ▶ **Uniqueness:** The leader can proceed with elevated permissions
- ▶ **Agreement:** Participant interaction can depend on known leader

We consider process-level implementation details

Goal: Mechanised functional correctness proofs for leader election implementations alongside other programming paradigms

Leader Election

Given a set of actors, leader election is an algorithm that satisfies:

- ▶ **Uniqueness:** There is exactly one actor that considers itself as leader
- ▶ **Agreement:** All other actors know who the leader is
- ▶ **Termination:** The algorithm finishes in finite time

We lift the properties to functional correctness as:

- ▶ **Uniqueness:** The leader can proceed with elevated permissions
- ▶ **Agreement:** Participant interaction can depend on known leader

We consider process-level implementation details

Goal: Mechanised functional correctness proofs for leader election implementations alongside other programming paradigms

Key idea: Mixed choice session types meets separation logic

Leader Election

Given a set of actors, leader election is an algorithm that satisfies:

- ▶ **Uniqueness:** There is exactly one actor that considers itself as leader <—
- ▶ **Agreement:** All other actors know who the leader is
- ▶ **Termination:** The algorithm finishes in finite time

We lift the properties to functional correctness as:

- ▶ **Uniqueness:** The leader can proceed with elevated permissions <—
- ▶ **Agreement:** Participant interaction can depend on known leader

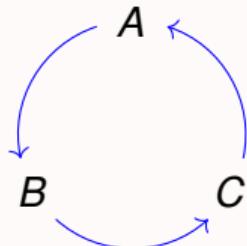
We consider process-level implementation details

Goal: Mechanised functional correctness proofs for leader election implementations alongside other programming paradigms

Key idea: Mixed choice session types meets separation logic

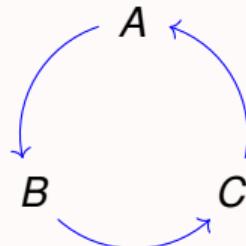
Mixed Choice in π -Calculus

Mixed Choice: simultaneously trying to send to- and receive from multiple parties



Mixed Choice in π -Calculus

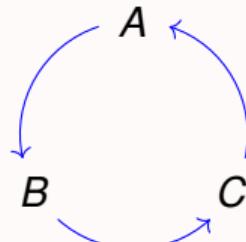
Mixed Choice: simultaneously trying to send to- and receive from multiple parties



$$\begin{aligned} e_A &:= ! [B] \langle v_1 \rangle . e_{A1} + ? [C] (x_3) . e_{A2} \\ || \quad e_B &:= ! [C] \langle v_2 \rangle . e_{B1} + ? [A] (x_1) . e_{B2} \\ || \quad e_C &:= ! [A] \langle v_3 \rangle . e_{C1} + ? [B] (x_2) . e_{C2} \end{aligned}$$

Mixed Choice in π -Calculus

Mixed Choice: simultaneously trying to send to- and receive from multiple parties

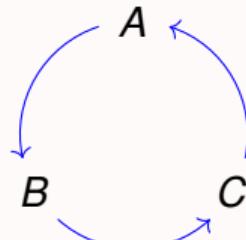


$$\begin{aligned} e_A &:= !B\langle v_1 \rangle . e_{A1} + ?C(x_3) . e_{A2} \\ || \quad e_B &:= !C\langle v_2 \rangle . e_{B1} + ?A(x_1) . e_{B2} \quad \Rightarrow \\ || \quad e_C &:= !A\langle v_3 \rangle . e_{C1} + ?B(x_2) . e_{C2} \end{aligned}$$

$$e_{A1} \parallel e_{B2}[v_1/x_1] \parallel e_C \quad e_A \parallel e_{B1} \parallel e_{C2}[v_2/x_2] \quad e_{A2}[v_3/x_3] \parallel e_B \parallel e_{C1}$$

Mixed Choice in π -Calculus

Mixed Choice: simultaneously trying to send to- and receive from multiple parties



$$\begin{aligned} e_A &:= !B\langle v_1 \rangle . e_{A1} + ?C(x_3) . e_{A2} \\ || \quad e_B &:= !C\langle v_2 \rangle . e_{B1} + ?A(x_1) . e_{B2} \quad \Rightarrow \\ || \quad e_C &:= !A\langle v_3 \rangle . e_{C1} + ?B(x_2) . e_{C2} \end{aligned}$$

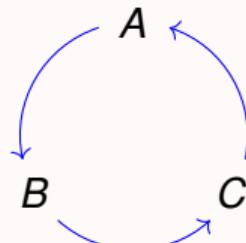
$$e_{A1} \parallel e_{B2}[v_1/x_1] \parallel e_C \quad e_A \parallel e_{B1} \parallel e_{C2}[v_2/x_2] \quad e_{A2}[v_3/x_3] \parallel e_B \parallel e_{C1}$$

Suitable for describing and verifying safety of leader election

- ▶ Mixed choice allows “breaking the symmetry” of processes [Palamidessi’13]
- ▶ Mixed choice session types allows scalable verification [Peters & Yoshida’24]

Mixed Choice in π -Calculus

Mixed Choice: simultaneously trying to send to- and receive from multiple parties



$$\begin{aligned} e_A &:= !B\langle v_1 \rangle . e_{A1} + ?C(x_3) . e_{A2} \\ || \quad e_B &:= !C\langle v_2 \rangle . e_{B1} + ?A(x_1) . e_{B2} \quad \Rightarrow \\ || \quad e_C &:= !A\langle v_3 \rangle . e_{C1} + ?B(x_2) . e_{C2} \end{aligned}$$

$$e_{A1} \parallel e_{B2}[v_1/x_1] \parallel e_C \quad e_A \parallel e_{B1} \parallel e_{C2}[v_2/x_2] \quad e_{A2}[v_3/x_3] \parallel e_B \parallel e_{C1}$$

Suitable for describing and verifying safety of leader election

- ▶ Mixed choice allows “breaking the symmetry” of processes [Palamidessi’13]
- ▶ Mixed choice session types allows scalable verification [Peters & Yoshida’24]

Problem: No work on functional correctness nor mechanisation of mixed choice

Functional Correctness of Message Passing

Actris: Separation logic for message passing in Iris mechanised in Rocq

- ▶ $\vdash \text{wp } e \{v. \text{True}\} \Rightarrow \text{safe}(e)$

Functional Correctness of Message Passing

Actris: Separation logic for message passing in Iris mechanised in Rocq

- ▶ $\vdash \text{wp } e \{v. \text{True}\} \Rightarrow \text{safe}(e)$

Start with an OCaml-like language with shared memory and concurrency

- ▶ $e := () | z | \ell | e; e | \text{let } x := e \text{ in } e | \text{ref } e | \text{free } e | \text{fork } \{e\} | \dots$

Functional Correctness of Message Passing

Actris: Separation logic for message passing in Iris mechanised in Rocq

- ▶ $\vdash \text{wp } e \{v. \text{True}\} \Rightarrow \text{safe}(e)$

Start with an OCaml-like language with shared memory and concurrency

- ▶ $e := () | z | \ell | e; e | \text{let } x := e \text{ in } e | \text{ref } e | \text{free } e | \text{fork } \{e\} | \dots$

Add or implement GV-style endpoints-as-terms message passing primitives

- ▶ **new_chan** (), **c.send**(v), **c.recv**()

Functional Correctness of Message Passing

Actris: Separation logic for message passing in Iris mechanised in Rocq

- ▶ $\vdash \text{wp } e \{v. \text{True}\} \Rightarrow \text{safe}(e)$

Start with an OCaml-like language with shared memory and concurrency

- ▶ $e := () | z | \ell | e; e | \text{let } x := e \text{ in } e | \text{ref } e | \text{free } e | \text{fork } \{e\} | \dots$

Add or implement GV-style endpoints-as-terms message passing primitives

- ▶ **new_chan** (), **c.send**(v), **c.recv**()

Prove verification rules for message passing primitives

- ▶ $\text{wp } \text{new_chan } () \{\dots\}$ $\text{wp } c.\text{send}(v) \{\dots\}$ $\text{wp } c.\text{recv}() \{\dots\}$

Functional Correctness of Message Passing

Actris: Separation logic for message passing in Iris mechanised in Rocq

- ▶ $\vdash \text{wp } e \{v. \text{True}\} \Rightarrow \text{safe}(e)$

Start with an OCaml-like language with shared memory and concurrency

- ▶ $e := () | z | \ell | e; e | \text{let } x := e \text{ in } e | \text{ref } e | \text{free } e | \text{fork } \{e\} | \dots$

Add or implement GV-style endpoints-as-terms message passing primitives

- ▶ **new_chan** (), $c.\text{send}(v)$, $c.\text{recv}()$

Prove verification rules for message passing primitives

- ▶ $\text{wp } \text{new_chan } () \{\dots\}$ $\text{wp } c.\text{send}(v) \{\dots\}$ $\text{wp } c.\text{recv}() \{\dots\}$

Prove WP's of programs using logic

Functional Correctness of Message Passing

Actris: Separation logic for message passing in Iris mechanised in Rocq

- ▶ $\vdash \text{wp } e \{v. \text{True}\} \Rightarrow \text{safe}(e)$

Start with an OCaml-like language with shared memory and concurrency

- ▶ $e := () | z | \ell | e; e | \text{let } x := e \text{ in } e | \text{ref } e | \text{free } e | \text{fork } \{e\} | \dots$

Add or implement GV-style endpoints-as-terms message passing primitives

- ▶ **new_chan** (), $c.\text{send}(v)$, $c.\text{recv}()$

Prove verification rules for message passing primitives

- ▶ $\text{wp } \text{new_chan } () \{\dots\}$ $\text{wp } c.\text{send}(v) \{\dots\}$ $\text{wp } c.\text{recv}() \{\dots\}$

Prove WP's of programs using logic

Previously: deadlock freedom, distributed systems, and multiparty

Functional Correctness of Message Passing

Actris: Separation logic for message passing in Iris mechanised in Rocq

- ▶ $\vdash \text{wp } e \{v. \text{True}\} \Rightarrow \text{safe}(e)$

Start with an OCaml-like language with shared memory and concurrency

- ▶ $e := () | z | \ell | e; e | \text{let } x := e \text{ in } e | \text{ref } e | \text{free } e | \text{fork } \{e\} | \dots$

Add or implement GV-style endpoints-as-terms message passing primitives

- ▶ **new_chan** (), $c.\text{send}(v)$, $c.\text{recv}()$

Prove verification rules for message passing primitives

- ▶ $\text{wp } \text{new_chan } () \{\dots\}$ $\text{wp } c.\text{send}(v) \{\dots\}$ $\text{wp } c.\text{recv}() \{\dots\}$

Prove WP's of programs using logic

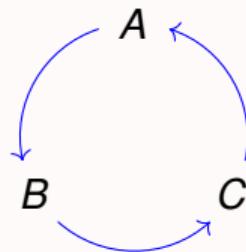
Previously: deadlock freedom, distributed systems, and multiparty

Problem: Not much work on mixed choice in non π -calculus settings

- ▶ “Mixed choice is a really difficult mechanism to implement” [Palamidessi’13]

Mixed Choice in Non π -Calculus

Per-thread semantics: Threads take turns (arbitrarily) reducing individually

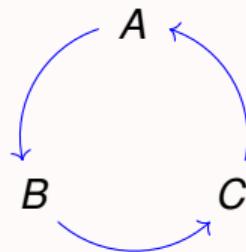


$$\begin{aligned} e_A &:= !B\langle v_1 \rangle.e_{A1} + ?C(x_3).e_{A2} \\ || \quad e_B &:= !C\langle v_2 \rangle.e_{B1} + ?A(x_1).e_{B2} \quad \Rightarrow^* \\ || \quad e_C &:= !A\langle v_3 \rangle.e_{C1} + ?B(x_2).e_{C2} \end{aligned}$$

$$e_{A1} \parallel e_{B2}[v_1/x_1] \parallel e_C \quad e_A \parallel e_{B1} \parallel e_{C2}[v_2/x_2] \quad e_{A2}[v_3/x_3] \parallel e_B \parallel e_{C1}$$

Mixed Choice in Non π -Calculus

Per-thread semantics: Threads take turns (arbitrarily) reducing individually



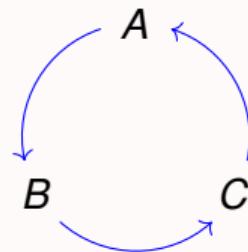
$$\begin{aligned} e_A &:= !B\langle v_1 \rangle.e_{A1} + ?C(x_3).e_{A2} \\ || \quad e_B &:= !C\langle v_2 \rangle.e_{B1} + ?A(x_1).e_{B2} \quad \Rightarrow^* \\ || \quad e_C &:= !A\langle v_3 \rangle.e_{C1} + ?B(x_2).e_{C2} \end{aligned}$$

$$e_{A1} \parallel e_{B2}[v_1/x_1] \parallel e_C \quad e_A \parallel e_{B1} \parallel e_{C2}[v_2/x_2] \quad e_{A2}[v_3/x_3] \parallel e_B \parallel e_{C1}$$

Non-collaborative concurrency: Threads cannot impact scheduling of others

Mixed Choice in Non π -Calculus

Per-thread semantics: Threads take turns (arbitrarily) reducing individually



$$\begin{aligned} e_A &:= !B\langle v_1 \rangle.e_{A1} + ?C(x_3).e_{A2} \\ || \quad e_B &:= !C\langle v_2 \rangle.e_{B1} + ?A(x_1).e_{B2} \quad \Rightarrow^* \\ || \quad e_C &:= !A\langle v_3 \rangle.e_{C1} + ?B(x_2).e_{C2} \end{aligned}$$

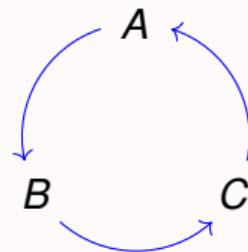
$$e_{A1} \parallel e_{B2}[v_1/x_1] \parallel e_C \quad e_A \parallel e_{B1} \parallel e_{C2}[v_2/x_2] \quad e_{A2}[v_3/x_3] \parallel e_B \parallel e_{C1}$$

Non-collaborative concurrency: Threads cannot impact scheduling of others

- ▶ Threads instead take turns trying to handshake until success

Mixed Choice in Non π -Calculus

Per-thread semantics: Threads take turns (arbitrarily) reducing individually



$$\begin{aligned} e_A &:= !B\langle v_1 \rangle.e_{A1} + ?C(x_3).e_{A2} \\ || \quad e_B &:= !C\langle v_2 \rangle.e_{B1} + ?A(x_1).e_{B2} \quad \Rightarrow^* \\ || \quad e_C &:= !A\langle v_3 \rangle.e_{C1} + ?B(x_2).e_{C2} \end{aligned}$$

$$e_{A1} \parallel e_{B2}[v_1/x_1] \parallel e_C \quad e_A \parallel e_{B1} \parallel e_{C2}[v_2/x_2] \quad e_{A2}[v_3/x_3] \parallel e_B \parallel e_{C1}$$

Non-collaborative concurrency: Threads cannot impact scheduling of others

- ▶ Threads instead take turns trying to handshake until success

Risk of repeatedly missing each other

- ▶ Although we have almost-sure termination under uniform scheduling

Contributions

Implementation of Mixed Choice Multiparty Channels in shared memory

- ▶ With safe synchronisation between participants

Contributions

Implementation of Mixed Choice Multiparty Channels in shared memory

- ▶ With safe synchronisation between participants

Mixed Choice Multiparty Dependent Separation Protocols (MCMDSPs)

- ▶ Rich protocols for describing mixed choice multiparty message passing

Contributions

Implementation of Mixed Choice Multiparty Channels in shared memory

- ▶ With safe synchronisation between participants

Mixed Choice Multiparty Dependent Separation Protocols (MCMDSps)

- ▶ Rich protocols for describing mixed choice multiparty message passing

Mixtris separation logic

- ▶ Bottom-up protocol consistency proven in separation logic
- ▶ Logic for verifying mixed choice multiparty communication via MCMDSps
- ▶ Verification rules proven against channel implementation

Contributions

Implementation of Mixed Choice Multiparty Channels in shared memory

- ▶ With safe synchronisation between participants

Mixed Choice Multiparty Dependent Separation Protocols (MCMDSps)

- ▶ Rich protocols for describing mixed choice multiparty message passing

Mixtris separation logic

- ▶ Bottom-up protocol consistency proven in separation logic
- ▶ Logic for verifying mixed choice multiparty communication via MCMDSps
- ▶ Verification rules proven against channel implementation

Verification suite of mixed choice multiparty programs

- ▶ Threeway election
- ▶ Chang and Roberts's ring leader election algorithm

Contributions

Implementation of Mixed Choice Multiparty Channels in shared memory

- ▶ With safe synchronisation between participants

Mixed Choice Multiparty Dependent Separation Protocols (MCMDS_Ps)

- ▶ Rich protocols for describing mixed choice multiparty message passing

Mixtris separation logic

- ▶ Bottom-up protocol consistency proven in separation logic
- ▶ Logic for verifying mixed choice multiparty communication via MCMDS_Ps
- ▶ Verification rules proven against channel implementation

Verification suite of mixed choice multiparty programs

- ▶ Threeway election
- ▶ Chang and Roberts's ring leader election algorithm

Full mechanisation in Rocq

- ▶ With automation support

Contributions

Implementation of Mixed Choice Multiparty Channels in shared memory <—

- ▶ With safe synchronisation between participants

Mixed Choice Multiparty Dependent Separation Protocols (MCMDSGs) <—

- ▶ Rich protocols for describing mixed choice multiparty message passing

Mixtris separation logic <—

- ▶ Bottom-up protocol consistency proven in separation logic
- ▶ Logic for verifying mixed choice multiparty communication via MCMDSGs
- ▶ Verification rules proven against channel implementation

Verification suite of mixed choice multiparty programs

- ▶ Threeway election <—
- ▶ Chang and Roberts's ring leader election algorithm

Full mechanisation in Rocq

- ▶ With automation support

Implementation of Mixed Choice

Mixed Choice Channel Endpoint Abstraction

Mixed choice channel endpoint API:

- `new_chan(n)`** Creates a mixed choice multiparty channel with n parties, returning a tuple $(c_0, \dots, c_{(n-1)})$ of endpoints
- `$c_i[j].try_send(v)$`** Momentarily tries to send value v via endpoint c_i to party j ; returns boolean signifying success
- `$c_i[j].try_recv()$`** Momentarily tries to receive a value via endpoint c_i to party j ; returns option of result

Mixed Choice Channel Endpoint Abstraction

Mixed choice channel endpoint API:

- `new_chan(n)`** Creates a mixed choice multiparty channel with n parties, returning a tuple $(c_0, \dots, c_{(n-1)})$ of endpoints
- `$c_i[j].try_send(v)$`** Momentarily tries to send value v via endpoint c_i to party j ; returns boolean signifying success
- `$c_i[j].try_recv()$`** Momentarily tries to receive a value via endpoint c_i to party j ; returns option of result

Encoding of simultaneous binary choice:

```
send_recv  $c i j v \triangleq$  if  $c[i].try\_send(v)$  then none  
                        else match  $c[j].try\_recv()$  with  
                        | some  $x \Rightarrow$  some  $x$   
                        | none    $\Rightarrow$  send_recv  $c i j v$   
                        end
```

Implementation of Mixed Choice Channel

Matrix of **synchronisation cells**, where i,j is used to send from i to j and vice versa

Implementation of Mixed Choice Channel

Matrix of **synchronisation cells**, where i,j is used to send from i to j and vice versa:

	0	...	i	...	j	...	n-1
0	-	...	$0 \rightarrow i$...	$0 \rightarrow j$...	$0 \rightarrow n-1$
...
i	$i \rightarrow 0$...	-	...	$i \rightarrow j$...	$i \rightarrow n-1$
...
j	$j \rightarrow 0$...	$j \rightarrow i$...	-	...	$j \rightarrow n-1$
...
n-1	$n-1 \rightarrow 0$...	$n-1 \rightarrow i$...	$n-1 \rightarrow j$...	-

Implementation of Mixed Choice Channel

Matrix of **synchronisation cells**, where i,j is used to send from i to j and vice versa:

	0	...	i	...	j	...	n-1
0	-	...	$0 \rightarrow i$...	$0 \rightarrow j$...	$0 \rightarrow n-1$
...
i	$i \rightarrow 0$...	-	...	$i \rightarrow j$...	$i \rightarrow n-1$
...
j	$j \rightarrow 0$...	$j \rightarrow i$...	-	...	$j \rightarrow n-1$
...
n-1	$n-1 \rightarrow 0$...	$n-1 \rightarrow i$...	$n-1 \rightarrow j$...	-

Channel endpoint c_i : Tuple of shared reference to matrix and own id i

Implementation of Mixed Choice Channel

Matrix of **synchronisation cells**, where i,j is used to send from i to j and vice versa:

	0	...	i	...	j	...	n-1
0	-	...	$0 \rightarrow i$...	$0 \rightarrow j$...	$0 \rightarrow n-1$
...
i	$i \rightarrow 0$...	-	...	$i \rightarrow j$...	$i \rightarrow n-1$
...
j	$j \rightarrow 0$...	$j \rightarrow i$...	-	...	$j \rightarrow n-1$
...
n-1	$n-1 \rightarrow 0$...	$n-1 \rightarrow i$...	$n-1 \rightarrow j$...	-

Channel endpoint c_i : Tuple of shared reference to matrix and own id i

- ▶ Send/Recv look up and use corresponding synchronisation cell in matrix

Implementation of Mixed Choice Channel

Matrix of **synchronisation cells**, where i,j is used to send from i to j and vice versa:

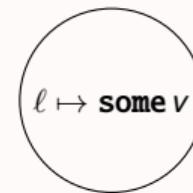
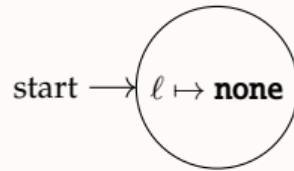
	0	...	i	...	j	...	n-1
0	-	...	$0 \rightarrow i$...	$0 \rightarrow j$...	$0 \rightarrow n-1$
...
i	$i \rightarrow 0$...	-	...	$i \rightarrow j$...	$i \rightarrow n-1$
...
j	$j \rightarrow 0$...	$j \rightarrow i$...	-	...	$j \rightarrow n-1$
...
n-1	$n-1 \rightarrow 0$...	$n-1 \rightarrow i$...	$n-1 \rightarrow j$...	-

Channel endpoint c_i : Tuple of shared reference to matrix and own id i

- ▶ Send/Recv look up and use corresponding synchronisation cell in matrix
- ▶ Synchronisation cells ensure appropriate synchronisation guarantees

Synchronisation Cells

Synchronisation cells: Reference ℓ with put and get where

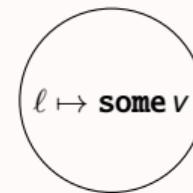
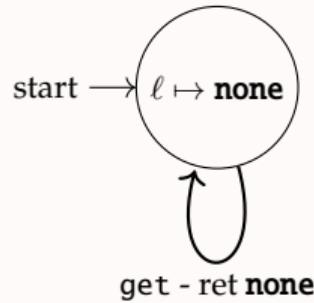


Synchronisation guarantees:

- ▶ Success when sender commits, receiver commits, sender observes
- ▶ Failure when sender commits, sender aborts, receiver tries to commit
- ▶ Guarantees synchronicity; sender/receiver both fail or succeed together

Synchronisation Cells

Synchronisation cells: Reference ℓ with put and get where

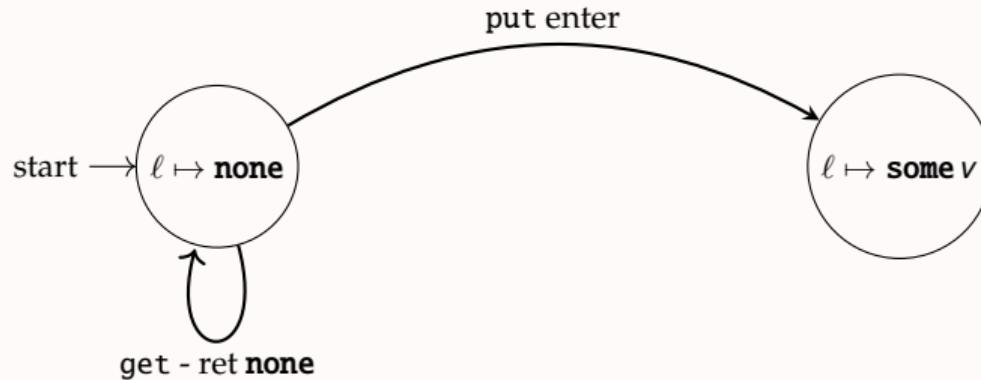


Synchronisation guarantees:

- ▶ Success when sender commits, receiver commits, sender observes
- ▶ Failure when sender commits, sender aborts, receiver tries to commit
- ▶ Guarantees synchronicity; sender/receiver both fail or succeed together

Synchronisation Cells

Synchronisation cells: Reference ℓ with put and get where

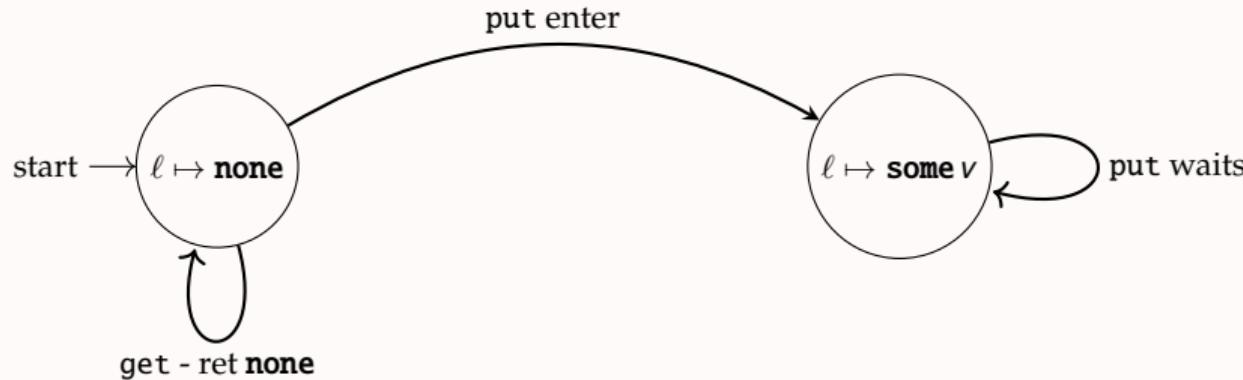


Synchronisation guarantees:

- ▶ Success when sender commits, receiver commits, sender observes
- ▶ Failure when sender commits, sender aborts, receiver tries to commit
- ▶ Guarantees synchronicity; sender/receiver both fail or succeed together

Synchronisation Cells

Synchronisation cells: Reference ℓ with put and get where

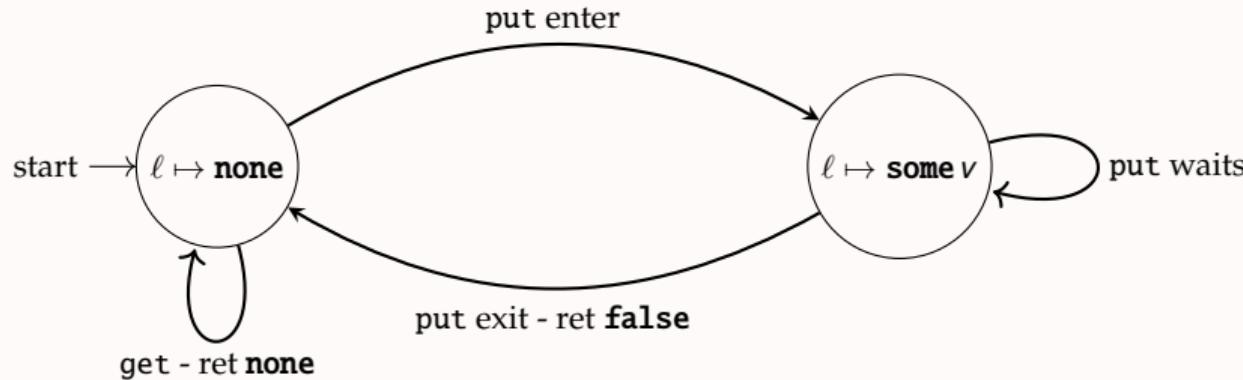


Synchronisation guarantees:

- ▶ Success when sender commits, receiver commits, sender observes
- ▶ Failure when sender commits, sender aborts, receiver tries to commit
- ▶ Guarantees synchronicity; sender/receiver both fail or succeed together

Synchronisation Cells

Synchronisation cells: Reference ℓ with put and get where

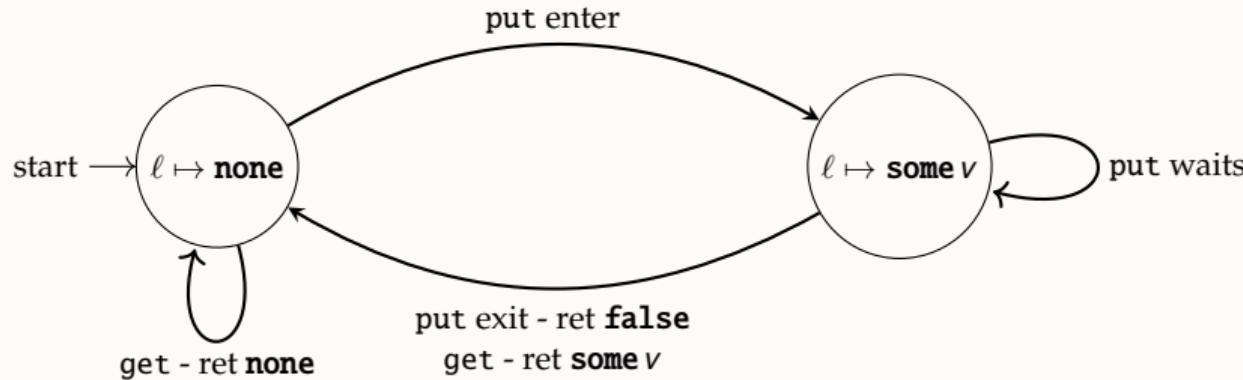


Synchronisation guarantees:

- ▶ Success when sender commits, receiver commits, sender observes
- ▶ Failure when sender commits, sender aborts, receiver tries to commit
- ▶ Guarantees synchronicity; sender/receiver both fail or succeed together

Synchronisation Cells

Synchronisation cells: Reference ℓ with put and get where

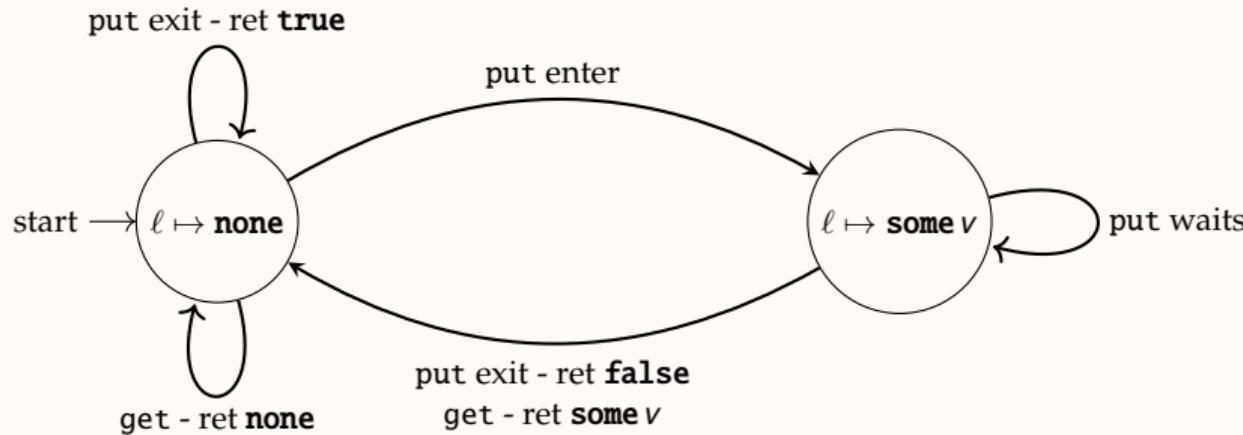


Synchronisation guarantees:

- ▶ Success when sender commits, receiver commits, sender observes
- ▶ Failure when sender commits, sender aborts, receiver tries to commit
- ▶ Guarantees synchronicity; sender/receiver both fail or succeed together

Synchronisation Cells

Synchronisation cells: Reference ℓ with put and get where



Synchronisation guarantees:

- ▶ Success when sender commits, receiver commits, sender observes
- ▶ Failure when sender commits, sender aborts, receiver tries to commit
- ▶ Guarantees synchronicity; sender/receiver both fail or succeed together

Implementation of Threeway Election

Example of encoded simultaneous binary choice:

```
send_recv c i j v  $\triangleq$  if  $c[i].try\_send(v)$  then none  
else match  $c[j].try\_recv()$  with  
| some  $x \Rightarrow$  some  $x$   
| none  $\Rightarrow$  send_recv  $c i j v$   
end
```

Implementation of threeway election: (where A:=0, B:=1, C:=2)

```
threeway_election_example  $\triangleq$   
let  $\ell := \text{ref } 42$  in  
let  $(c_A, c_B, c_C) := \text{new\_chan}(3)$  in  
fork {match send_recv  $c_B C A ()$  with none  $\Rightarrow ()$ ; some _  $\Rightarrow \text{free } \ell \text{ end}$ } ;  
fork {match send_recv  $c_C A B ()$  with none  $\Rightarrow ()$ ; some _  $\Rightarrow \text{free } \ell \text{ end}$ } ;  
fork {match send_recv  $c_A B C ()$  with none  $\Rightarrow ()$ ; some _  $\Rightarrow \text{free } \ell \text{ end}$ }
```

Mixed Choice Multiparty Dependent Separation Protocols

Leader Uniqueness

```
threeway_election_example  $\triangleq$ 
  let  $\ell := \text{ref } 42$  in
  let  $(c_A, c_B, c_C) := \text{new\_chan}(3)$  in
  fork {match send_recv  $c_B$  C A () with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_C$  A B () with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_A$  B C () with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ }
```

Goal: Prove leader uniqueness (no double-free)

Leader Uniqueness

```
threeway_election_example  $\triangleq$ 
  let  $\ell := \text{ref } 42$  in
  let  $(c_A, c_B, c_C) := \text{new\_chan}(3)$  in
  fork {match send_recv  $c_B C A ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_C A B ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_A B C ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ }
```

Goal: Prove leader uniqueness (no double-free)

Mixed choice multiparty session types:

$$c_A : ! [B]. \mathbf{end} + ? [C]. \mathbf{end} \quad c_B : ! [C]. \mathbf{end} + ? [A]. \mathbf{end} \quad c_C : ! [A]. \mathbf{end} + ? [B]. \mathbf{end}$$

Leader Uniqueness

```
threeway_election_example  $\triangleq$ 
let  $\ell := \text{ref } 42$  in
let  $(c_A, c_B, c_C) := \text{new\_chan}(3)$  in
fork {match send_recv  $c_B$  C A () with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
fork {match send_recv  $c_C$  A B () with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
fork {match send_recv  $c_A$  B C () with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ }
```

Goal: Prove leader uniqueness (no double-free)

Mixed choice multiparty session types:

$$c_A : ! [B]. \mathbf{end} + ? [C]. \mathbf{end} \quad c_B : ! [C]. \mathbf{end} + ? [A]. \mathbf{end} \quad c_C : ! [A]. \mathbf{end} + ? [B]. \mathbf{end}$$

Problem: No way of associating elected leader with privilege to free

Mixed Choice Multiparty Dependent Separation Protocols

Enriching protocols with separation logic resources

- ▶ $![B]. \mathbf{end} + ?[C]. \mathbf{end} \rightarrow ! [B] . \mathbf{end} + ?[C] \{\ell \mapsto 42\}. \mathbf{end}$

Mixed Choice Multiparty Dependent Separation Protocols

Enriching protocols with separation logic resources

$$\triangleright \text{!}[B]. \mathbf{end} + ?[C]. \mathbf{end} \rightarrow \text{!}[B]. \mathbf{end} + ?[C] \{\ell \mapsto 42\}. \mathbf{end}$$

Mixed choice multiparty dependent separation protocols (OBS: Simplified)

$$\triangleright p ::= \text{!}[i] \{P\}. p \mid ?[i] \{P\}. p \mid p + q \mid \mathbf{end}$$

Mixed Choice Multiparty Dependent Separation Protocols

Enriching protocols with separation logic resources

$$\triangleright \mathbf{!}[B].\mathbf{end} + \mathbf{?}[C].\mathbf{end} \rightarrow \mathbf{!}[B].\mathbf{end} + \mathbf{?}[C]\{\ell \mapsto 42\}.\mathbf{end}$$

Mixed choice multiparty dependent separation protocols (OBS: Simplified)

$$\triangleright p ::= \mathbf{!}[i]\{P\}.p \mid \mathbf{?}[i]\{P\}.p \mid p + q \mid \mathbf{end}$$

Sufficient for describing threeway leader process

```
match send_recv c_A B C () with none => (); some _ => free l end
```

Mixed Choice Multiparty Dependent Separation Protocols

Enriching protocols with separation logic resources

$$\triangleright \mathbf{!}[B].\mathbf{end} + \mathbf{?}[C].\mathbf{end} \rightarrow \mathbf{!}[B].\mathbf{end} + \mathbf{?}[C]\{\ell \mapsto 42\}.\mathbf{end}$$

Mixed choice multiparty dependent separation protocols (OBS: Simplified)

$$\triangleright p ::= \mathbf{!}[i]\{P\}.p \mid \mathbf{?}[i]\{P\}.p \mid p + q \mid \mathbf{end}$$

Sufficient for describing threeway leader process

```
match send_recv c_A B C () with none => (); some _ => free l end
```

Remaining challenges:

- ▶ How to guarantee consistent global communication?
- ▶ How to apply protocols to verify program?

The Mixtris Separation Logic

Protocol Consistency

Remaining challenge: How to guarantee consistent global communication?

```
threeway_election_example  $\triangleq$ 
  let  $\ell := \text{ref } 42$  in
  let  $(c_A, c_B, c_C) := \text{new\_chan}(3)$  in
  fork {match send_recv  $c_B C A ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_C A B ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_A B C ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ }
```

Protocol Consistency

Remaining challenge: How to guarantee consistent global communication?

```
threeway_election_example  $\triangleq$ 
  let  $\ell := \text{ref} 42$  in
  let  $(c_A, c_B, c_C) := \text{new\_chan}(3)$  in
  fork {match send_recv  $c_B C A ()$  with none  $\Rightarrow ()$ ; some _  $\Rightarrow \text{free } \ell \text{ end}} ;
  fork {match send_recv  $c_C A B ()$  with none  $\Rightarrow ()$ ; some _  $\Rightarrow \text{free } \ell \text{ end}} ;
  fork {match send_recv  $c_A B C ()$  with none  $\Rightarrow ()$ ; some _  $\Rightarrow \text{free } \ell \text{ end}}$$$ 
```

Prior work: Syntactic duality

$$\begin{aligned} c_A &: ! [B]. \mathbf{end} + ? [C]. \mathbf{end} \\ c_B &: ! [C]. \mathbf{end} + ? [A]. \mathbf{end} \\ c_C &: ! [A]. \mathbf{end} + ? [B]. \mathbf{end} \end{aligned}$$

Protocol Consistency

Remaining challenge: How to guarantee consistent global communication?

```
threeway_election_example  $\triangleq$ 
  let  $\ell := \text{ref } 42$  in
  let  $(c_A, c_B, c_C) := \text{new\_chan}(3)$  in
  fork {match send_recv  $c_B C A ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_C A B ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_A B C ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ }
```

Prior work: Syntactic duality

$$\begin{aligned} c_A &: ! [B]. \mathbf{end} + ? [C]. \mathbf{end} \\ c_B &: ! [C]. \mathbf{end} + ? [A]. \mathbf{end} \\ c_C &: ! [A]. \mathbf{end} + ? [B]. \mathbf{end} \end{aligned}$$

This work:

$$\begin{aligned} c_A &\mapsto ! [B] . \mathbf{end} + ? [C] \{ \ell \mapsto 42 \} . \mathbf{end} \\ c_B &\mapsto ! [C] . \mathbf{end} + ? [A] \{ \ell \mapsto 42 \} . \mathbf{end} \\ c_C &\mapsto ! [A] . \mathbf{end} + ? [B] \{ \ell \mapsto 42 \} . \mathbf{end} \end{aligned}$$

Protocol Consistency

Remaining challenge: How to guarantee consistent global communication?

```
threeway_election_example  $\triangleq$ 
  let  $\ell := \text{ref } 42$  in
  let  $(c_A, c_B, c_C) := \text{new\_chan}(3)$  in
  fork {match send_recv  $c_B C A ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_C A B ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_A B C ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ }
```

Prior work: Syntactic duality

$$\begin{aligned} c_A &: ! [B]. \mathbf{end} + ? [C]. \mathbf{end} \\ c_B &: ! [C]. \mathbf{end} + ? [A]. \mathbf{end} \\ c_C &: ! [A]. \mathbf{end} + ? [B]. \mathbf{end} \end{aligned}$$

This work: Contextual semantic duality

$$\begin{aligned} c_A &\mapsto ! [B] . \mathbf{end} + ? [C] \{ \ell \mapsto 42 \} . \mathbf{end} \\ c_B &\mapsto ! [C] . \mathbf{end} + ? [A] \{ \ell \mapsto 42 \} . \mathbf{end} \\ c_C &\mapsto ! [A] . \mathbf{end} + ? [B] \{ \ell \mapsto 42 \} . \mathbf{end} \end{aligned}$$

Protocol Consistency

Remaining challenge: How to guarantee consistent global communication?

```
threeway_election_example  $\triangleq$ 
  let  $\ell := \text{ref } 42$  in
  let  $(c_A, c_B, c_C) := \text{new\_chan}(3)$  in
  fork {match send_recv  $c_B C A ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_C A B ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_A B C ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ }
```

Prior work: Syntactic duality

$$\begin{aligned} c_A &: !B.\text{end} + ?C.\text{end} \\ c_B &: !C.\text{end} + ?A.\text{end} \\ c_C &: !A.\text{end} + ?B.\text{end} \end{aligned}$$

This work: Contextual semantic duality

$$\begin{aligned} c_A &\mapsto !B.\text{end} + ?C \{ \ell \mapsto 42 \}.\text{end} \\ c_B &\mapsto !C.\text{end} + ?A \{ \ell \mapsto 42 \}.\text{end} \\ c_C &\mapsto !A.\text{end} + ?B \{ \ell \mapsto 42 \}.\text{end} \end{aligned}$$

Key Idea: Define and prove consistency via separation logic!

Protocol Consistency

We define a set of protocols p_0, \dots, p_k to be CONSISTENT (p_0, \dots, p_k) whenever, for any synchronised pair i, j , given the resources of i , and any prior resources

1. Provide the resources of j
2. Prove protocol consistency where i and j are updated to their respective tails

Repeat until no more synchronised pairs exist.

Protocol Consistency

We define a set of protocols p_0, \dots, p_k to be CONSISTENT (p_0, \dots, p_k) whenever, for any synchronised pair i, j , given the resources of i , and any prior resources

1. Provide the resources of j
2. Prove protocol consistency where i and j are updated to their respective tails

Repeat until no more synchronised pairs exist.

Given $\ell \mapsto 42$, show consistency of

$$p_A := (! [B] . \mathbf{end}) + (?[C] \{\ell \mapsto 42\} . \mathbf{end})$$

$$p_B := (! [C] . \mathbf{end}) + (?[A] \{\ell \mapsto 42\} . \mathbf{end})$$

$$p_C := (! [A] . \mathbf{end}) + (?[B] \{\ell \mapsto 42\} . \mathbf{end})$$

p_A sends to p_B	p_B sends to p_C	p_C sends to c_A
$p_A := \mathbf{end}$	$p_A := p_A$	$p_A := \mathbf{end}$
$p_B := \mathbf{end}$	$p_B := \mathbf{end}$	$p_B := p_B$
$p_C := p_C$	$p_C := \mathbf{end}$	$p_C := \mathbf{end}$

Mixtris

$$\frac{W_{P\text{-NEW}} \quad k = n - 1 \quad \text{CONSISTENT } (p_0, \dots, p_k) \quad n > 0}{\text{wp } \mathbf{new_chan}(n) \{ (c_0, \dots, c_k). c_0 \rightarrow p_0 * \dots * c_k \rightarrow p_k \}^*}$$

Mixtris

$W_{P\text{-NEW}}$

$$\frac{k = n - 1 \quad \text{CONSISTENT } (p_0, \dots, p_k) \quad n > 0}{\text{wp } \mathbf{new_chan}(n) \{ (c_0, \dots, c_k). c_0 \rightarrowtail p_0 * \dots * c_k \rightarrowtail p_k \}}^*$$

$W_{P\text{-TRY-SEND}}$

$$\frac{c \rightarrowtail ![i] \{P\}. p \quad P}{\text{wp } c[i]. \mathbf{try_send}() \{ b. \mathbf{if } b \mathbf{then } c \rightarrowtail p \mathbf{else } c \rightarrowtail ![i] \{P\}. p * P \}}^*$$

Mixtris

$W_{P\text{-NEW}}$

$$\frac{k = n - 1 \quad \text{CONSISTENT } (p_0, \dots, p_k) \quad n > 0}{\text{wp } \mathbf{new_chan}(n) \{ (c_0, \dots, c_k). c_0 \rightarrowtail p_0 * \dots * c_k \rightarrowtail p_k \}}^*$$

CHAN-SUB

$$\frac{c \rightarrowtail p_1 \quad p_1 \sqsubseteq p_2}{c \rightarrowtail p_2}^*$$

$W_{P\text{-TRY-SEND}}$

$$\frac{c \rightarrowtail ![i] \{P\}.p \quad P}{\text{wp } c[i].\mathbf{try_send}() \{ b. \mathbf{if } b \mathbf{then } c \rightarrowtail p \mathbf{else } c \rightarrowtail ![i] \{P\}.p * P \}}^*$$

SUB-CHOICE-L

$$p + q \sqsubseteq p$$

Mixtris

$W_{P\text{-NEW}}$

$$\frac{k = n - 1 \quad \text{CONSISTENT } (p_0, \dots, p_k) \quad n > 0}{\text{wp } \mathbf{new_chan}(n) \{ (c_0, \dots, c_k). c_0 \rightarrow p_0 * \dots * c_k \rightarrow p_k \}}^*$$

CHAN-SUB

$$\frac{c \rightarrow p_1 \quad p_1 \sqsubseteq p_2}{c \rightarrow p_2}^*$$

$W_{P\text{-TRY-SEND}}$

$$\frac{c \rightarrow q \quad q \sqsubseteq ! [i] \{ P \}. p \quad P}{\text{wp } c[i]. \mathbf{try_send}() \{ b. \mathbf{if } b \mathbf{then } c \rightarrow p \mathbf{else } c \rightarrow q * P \}}^*$$

SUB-CHOICE-L

$$p + q \sqsubseteq p$$

Mixtris

$W_{P\text{-NEW}}$

$$\frac{k = n - 1 \quad \text{CONSISTENT } (p_0, \dots, p_k) \quad n > 0}{\text{wp } \mathbf{new_chan}(n) \{ (c_0, \dots, c_k). c_0 \rightarrow p_0 * \dots * c_k \rightarrow p_k \}}^*$$

CHAN-SUB

$$\frac{c \rightarrow p_1 \quad p_1 \sqsubseteq p_2}{c \rightarrow p_2}^*$$

$W_{P\text{-TRY-SEND}}$

$$\frac{c \rightarrow q \quad q \sqsubseteq ![i] \{P\}.p \quad P}{\text{wp } c[i].\mathbf{try_send}() \{ b. \mathbf{if } b \mathbf{then } c \rightarrow p \mathbf{else } c \rightarrow q * P \}}^*$$

$$\text{SUB-CHOICE-L} \\ p + q \sqsubseteq p$$

$W_{P\text{-TRY-RECV}}$

$$\frac{c \rightarrow q \quad q \sqsubseteq ?[i] \{P\}.p}{\text{wp } c[i].\mathbf{try_recv}() \left\{ \begin{array}{l} \mathbf{match } ov \mathbf{with} \\ | \mathbf{some } w \Rightarrow w = () * c \rightarrow p * P \\ | \mathbf{none } \Rightarrow c \rightarrow q \\ \mathbf{end} \end{array} \right\}}^*$$

Threeway Election - Verified

Threeway election program:

```
threeway_election_example  $\triangleq$ 
  let  $\ell := \text{ref } 42$  in
  let  $(c_A, c_B, c_C) := \text{new\_chan}(3)$  in
  fork {match send_recv  $c_B$   $C A ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell$  end} ;
  fork {match send_recv  $c_C$   $A B ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell$  end} ;
  fork {match send_recv  $c_A$   $B C ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell$  end}
```

Threeway Election - Verified

Threeway election program:

```
threeway_election_example  $\triangleq$ 
  let  $\ell := \text{ref } 42$  in
  let  $(c_A, c_B, c_C) := \text{new\_chan}(3)$  in
  fork {match send_recv  $c_B$   $C A ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_C$   $A B ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_A$   $B C ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ }
```

Protocols:

$$\begin{aligned} c_A &\mapsto (! [B] . \mathbf{end}) + (?[C] \{\ell \mapsto 42\} . \mathbf{end}) \\ c_B &\mapsto (! [C] . \mathbf{end}) + (?[A] \{\ell \mapsto 42\} . \mathbf{end}) \\ c_C &\mapsto (! [A] . \mathbf{end}) + (?[B] \{\ell \mapsto 42\} . \mathbf{end}) \end{aligned}$$

Threeway Election - Verified

Threeway election program:

```
threeway_election_example  $\triangleq$ 
  let  $\ell := \text{ref } 42$  in
  let  $(c_A, c_B, c_C) := \text{new\_chan}(3)$  in
  fork {match send_recv  $c_B$   $C A ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_C$   $A B ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ } ;
  fork {match send_recv  $c_A$   $B C ()$  with none  $\Rightarrow ()$ ; some_  $\Rightarrow \text{free } \ell \text{ end}$ }
```

Protocols:

$$\begin{aligned} c_A &\mapsto (! [B] . \mathbf{end}) + (?[C] \{\ell \mapsto 42\} . \mathbf{end}) \\ c_B &\mapsto (! [C] . \mathbf{end}) + (?[A] \{\ell \mapsto 42\} . \mathbf{end}) \\ c_C &\mapsto (! [A] . \mathbf{end}) + (?[B] \{\ell \mapsto 42\} . \mathbf{end}) \end{aligned}$$

Verified functional correctness!

Concluding Remarks

More in the Paper

Full mixed choice multiparty dependent separation protocols and rules

- ▶ Dependent binders, exchanged values, value-dependent branching, recursion
- ▶ $p ::= ! [i] (\vec{x} : \vec{\tau}) \langle v \rangle \{P\}.p \mid ?[i] (\vec{x} : \vec{\tau}) \langle v \rangle \{P\}.p \mid \mathbf{end} \mid p + q \mid \mu x.p$

Committed send/recv channel primitives

- ▶ $c[i].\mathbf{send}(v)$ and $c[i].\mathbf{recv}()$
- ▶ Seamlessly compose with uncommitted send/receive

Verification of Chang and Roberts's ring leader election

- ▶ Including verification of leader agreement

Language-agnostic verification interface for MCMDSPs

- ▶ Implementation and verification of synchronisation cells

Conclusion

Mixed choice remains non-trivial to implement

- ▶ Atomic receiver commit point was crucial for synchronicity
- ▶ Proper liveness contingent on collaborative concurrency
- ▶ Unclear how to achieve in a proper distributed system

Contextual protocol consistency useful for modelling leader election

- ▶ Resources often needs to be delegated on completion
- ▶ Resources does not necessarily enter the system during the protocol

Future Work

Semantic Multiparty Mixed Choice Session Type System

- ▶ Investigate correspondences with syntactic protocol consistency

More scalable proofs of protocol consistency

- ▶ Abstraction and modularity via separation logic
- ▶ Automation via Model Checking

Deadlock freedom and liveness guarantees

- ▶ Leverage connectivity graphs for multiparty communication

Mixtris for distributed systems

- ▶ Probabilistic implementations for mixed choice

$! [A] \langle \text{“Thanks”} \rangle \{\text{MixtrisOverview}\}.$
 $\mu rec. (?[A] (q : \text{Question i}) \langle q \rangle \{\text{AboutMultris } q\}.$
 $! [A] (a : \text{Answer}) \langle a \rangle \{\text{Insightful } q\ a\}. rec)$
 $+$
 $! [C] \langle \text{“Times up?”} \rangle. \text{end}$

Backup Slides

Chang and Roberts's Protocol

$$\begin{aligned} \text{cre_init_process_prot } & (i : \mathbb{N}) (P : \text{iProp}) (p : \mathbb{N} \rightarrow \text{iProto}) : \text{iProto} \triangleq \\ & ! [i_l] \langle \text{inl } i \rangle . \text{cre_process_prot } i P p + \\ & ? [i_r] (i' : \mathbb{N}) \langle \text{inl } i' \rangle . \left\{ \begin{array}{l} \text{if } i < i' \text{ then } ! [i_l] \langle \text{inl } i' \rangle . \text{cre_process_prot } i P p \\ \text{else if } i = i' \text{ then } ! [i_l] \langle \text{inr } i \rangle . \text{cre_process_prot } i P p \\ \text{else } ! [i_l] \langle \text{inl } i \rangle . \text{cre_process_prot } i P p \end{array} \right\} \end{aligned}$$

Chang and Roberts's Protocol

$\text{cre_init_process_prot } (i : \mathbb{N}) (P : \text{iProp}) (p : \mathbb{N} \rightarrow \text{iProto}) : \text{iProto} \triangleq$

$\quad ! [i_l] \langle \text{inl } i \rangle. \text{cre_process_prot } i P p +$

$\quad ? [i_r] (i' : \mathbb{N}) \langle \text{inl } i' \rangle. \left\{ \begin{array}{l} \text{if } i < i' \text{ then } ! [i_l] \langle \text{inl } i' \rangle. \text{cre_process_prot } i P p \\ \text{else if } i = i' \text{ then } ! [i_l] \langle \text{inr } i \rangle. \text{cre_process_prot } i P p \\ \text{else } ! [i_l] \langle \text{inl } i \rangle. \text{cre_process_prot } i P p \end{array} \right\}$

$\text{cre_process_prot } (i : \mathbb{N}) (P : \text{iProp}) (p : \mathbb{N} \rightarrow \text{iProto}) : \text{iProto} \triangleq \mu \text{rec.}$

$\quad \& [i_r] \left\{ \begin{array}{ll} \text{inl}(i' : \mathbb{N}) \langle i' \rangle & \Rightarrow \text{if } i < i' \text{ then } ! [i_l] \langle \text{inl } i' \rangle. \text{rec} \\ & \text{else if } i = i' \text{ then } ! [i_l] \langle \text{inr } i \rangle. \text{rec} \\ & \text{else } \text{rec} \\ \text{inr}(i' : \mathbb{N}) \langle i' \rangle \{i = i' \rightarrow P\} & \Rightarrow \text{if } i = i' \text{ then } p i' \\ & \text{else } ! [i_l] \langle \text{inr } i' \rangle. p i' \end{array} \right\}$

Implementation - Synchronisation Cells

```
new_sync () := ref none
```

```
wait c := match !c with
| none   => ()
| some _  => wait c
end
```

```
sync_put c v := c ← some v;
               wait c.
```

```
sync_get c :=
```

```
match Xchg c none with
| none   => sync_get c
| some v => v
end
```

```
sync_try_put c v :=
c ← some v;
match Xchg c none with
| none   => true
| some _  => false
end
```

```
sync_try_get c := Xchg c none
```

Implementation - Channels

Array of synchronisation cells

```
new_chan(n) := let m := new_matrix n n (new_sync()) in ((m, 0), ..., (m, n - 1))
```

$$c[j].\text{try_send}(v) := \text{let } (m, i) := c \text{ in} \quad \quad \quad c[j].\text{try_recv}() := \text{let } (m, i) := c \text{ in} \\ \qquad \qquad \qquad \text{sync_try_put } m_{i,j} \ v \qquad \qquad \qquad \qquad \qquad \text{sync_try_get } m_{j,i}$$

Mixtris Ghost Theory

We defined the our protocols via Iris's recursive domain equation solver and proved language-generic ghost theory rules based on Iris's ghost state machinery

PROTO-ALLOC

$$\frac{\text{CONSISTENT } \vec{p}}{\Rightarrow \exists \chi. \text{prot_ctx } \chi \mid \vec{p} \mid * \quad \begin{array}{c} \ast \\ i \mapsto p \in \vec{p} \end{array} \quad \text{prot_own } \chi \ i \ p} *$$

PROTO-LE

$$\frac{\text{prot_own } \chi \ i \ p_1 \quad p_1 \sqsubseteq p_2}{\text{prot_own } \chi \ i \ p_2} *$$

PROTO-STEP

$$\frac{\text{prot_ctx } \chi \ n \quad P_1 \quad \text{prot_own } \chi \ i \ (![j] \ {P_1}. \ p_1) \quad \text{prot_own } \chi \ j \ (?[i] \ {P_2}. \ p_2)}{\Rightarrow \triangleright \text{prot_ctx } \chi \ n * \text{prot_own } \chi \ i \ p_1 * \text{prot_own } \chi \ j \ p_2 * P_2} *$$

One can then define language-specific $c \rightarrow p$ and prove Hoare triple rules (such as W_P -TRY-SEND, W_P -TRY-RECV, and W_P -NEW) for an implementation using the ghost theory