

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

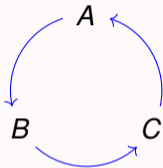
We consider process-level implementation details

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

Key idea: Mixed choice session types meets separation logic

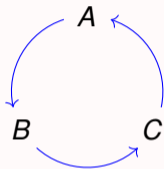
Mixed Choice in π -Calculus

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



Mixed Choice in π -Calculus

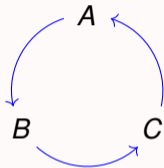
Mixed Choice: simultaneously trying to send to- and receive from multiple parties



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

Mixed Choice in π -Calculus

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

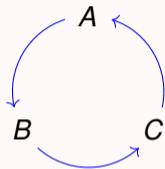


$$\begin{aligned} e_A &:= ![B]\langle v_1 \rangle . e_{A1} + ?[C](x_3) . e_{A2} \\ || e_B &:= ![C]\langle v_2 \rangle . e_{B1} + ?[A](x_1) . e_{B2} \\ || e_C &:= ![A]\langle v_3 \rangle . e_{C1} + ?[B](x_2) . e_{C2} \end{aligned} \quad \Rightarrow$$

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

Mixed Choice in π -Calculus

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



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

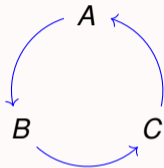
$$e_{A1} \parallel e_{B2}[v_1/x_1] \parallel e_C \quad e_A \parallel e_{B1} \parallel e_{C2}[v_2/x_2] \quad e_{A2}[v_3/x_3] \parallel e_B \parallel e_{C1}$$

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



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

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

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

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

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

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

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

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

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

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

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

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

Prove verification rules for message passing primitives

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

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

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

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

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

Prove verification rules for message passing primitives

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

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

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

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

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

Prove verification rules for message passing primitives

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

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

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

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

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

Prove verification rules for message passing primitives

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

Prove WP's of programs using logic

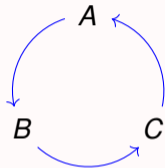
Previously: deadlock freedom, distributed systems, and multiparty

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

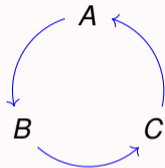


$$\begin{aligned} e_A &:= ![B]\langle v_1 \rangle . e_{A1} + ?[C](x_3) . e_{A2} \\ || e_B &:= ![C]\langle v_2 \rangle . e_{B1} + ?[A](x_1) . e_{B2} \\ || e_C &:= ![A]\langle v_3 \rangle . e_{C1} + ?[B](x_2) . e_{C2} \end{aligned} \quad \Rightarrow^*$$

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

Mixed Choice in Non π -Calculus

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



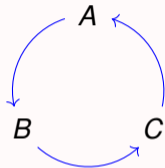
$$\begin{aligned} e_A &:= ![B]\langle v_1 \rangle . e_{A1} + ?[C](x_3) . e_{A2} \\ || e_B &:= ![C]\langle v_2 \rangle . e_{B1} + ?[A](x_1) . e_{B2} \\ || e_C &:= ![A]\langle v_3 \rangle . e_{C1} + ?[B](x_2) . e_{C2} \end{aligned} \Rightarrow^*$$

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

Non-collaborative concurrency: Threads cannot impact scheduling of others

Mixed Choice in Non π -Calculus

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



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

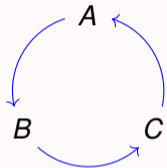
$$e_{A1} \parallel e_{B2}[v_1/x_1] \parallel e_C \quad e_A \parallel e_{B1} \parallel e_{C2}[v_2/x_2] \quad e_{A2}[v_3/x_3] \parallel e_B \parallel e_{C1}$$

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



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

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

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

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

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 (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

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 (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

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
- $c_i[j].\text{try_send}(v)$** Momentarily tries to send value v via endpoint c_i to party j ; returns boolean signifying success
- $c_i[j].\text{try_recv}()$** 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
- $c_i[j].\text{try_send}(v)$** Momentarily tries to send value v via endpoint c_i to party j ; returns boolean signifying success
- $c_i[j].\text{try_recv}()$** Momentarily tries to receive a value via endpoint c_i to party j ; returns option of result

Encoding of simultaneous binary choice:

$$\text{send_recv } c \ i \ j \ v \triangleq \text{if } c[i].\text{try_send}(v) \text{ then none} \\ \text{else match } c[j].\text{try_recv}() \text{ with} \\ \quad | \text{ some } x \Rightarrow \text{some } x \\ \quad | \text{ none } \Rightarrow \text{send_recv } c \ i \ j \ v \\ \text{end}$$

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

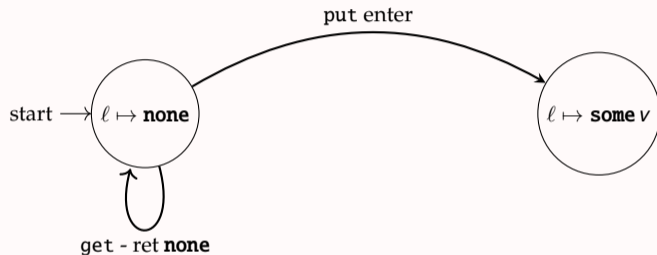


Synchronisation guarantees:

- ▶ 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

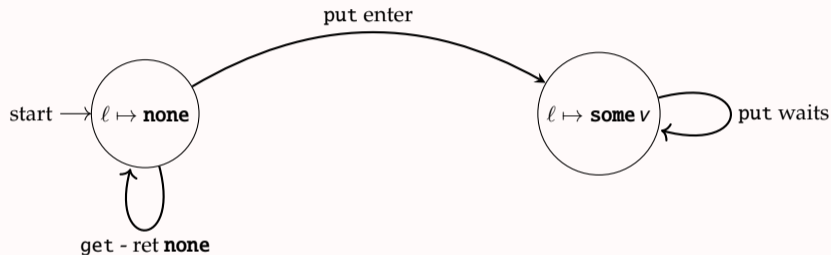


Synchronisation guarantees:

- ▶ 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

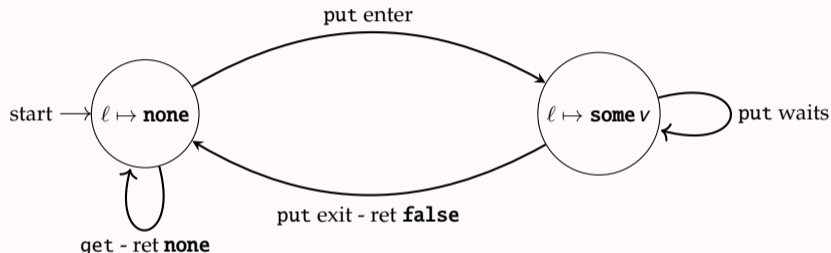


Synchronisation guarantees:

- ▶ 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

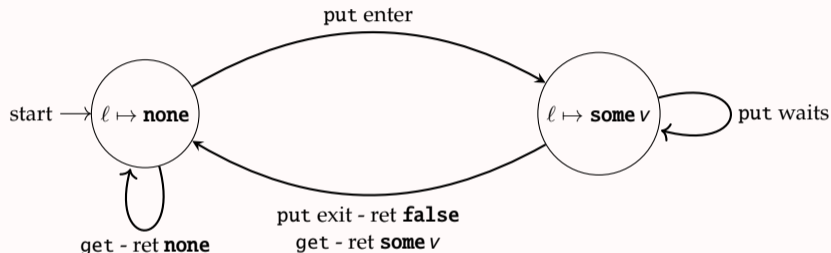


Synchronisation guarantees:

- ▶ 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

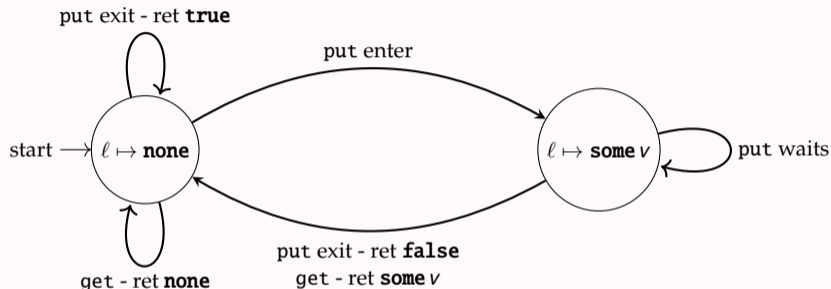


Synchronisation guarantees:

- ▶ 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 (cA, cB, cC) := new_chan(3) in  
  fork {match send_recv cB C A () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv cC A B () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv cA B C () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  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 _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_C$   $A$   $B$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_A$   $B$   $C$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  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 _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_C$   $A$   $B$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_A$   $B$   $C$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  end}
```

Goal: Prove leader uniqueness (no double-free)

Mixed choice multiparty session types:

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

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  $\_ \Rightarrow \text{free } \ell$  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}
```

Goal: Prove leader uniqueness (no double-free)

Mixed choice multiparty session types:

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

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

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

Mixed choice multiparty dependent separation protocols (OBS: Simplified)

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

Mixed Choice Multiparty Dependent Separation Protocols

Enriching protocols with separation logic resources

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

Mixed choice multiparty dependent separation protocols (OBS: Simplified)

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

Sufficient for describing threeway leader process

$\mathbf{match} \text{ send_recv } c_A B C () \mathbf{with} \text{ none } \Rightarrow (); \text{ some } _ \Rightarrow \mathbf{free} \ell \mathbf{end}$

Mixed Choice Multiparty Dependent Separation Protocols

Enriching protocols with separation logic resources

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

Mixed choice multiparty dependent separation protocols (OBS: Simplified)

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

Sufficient for describing threeway leader process

match send_recv $c_A B C ()$ **with** none $\Rightarrow ()$; some $_ \Rightarrow \text{free } \ell$ **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 _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_C$   $A$   $B$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_A$   $B$   $C$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  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 _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_C$   $A$   $B$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_A$   $B$   $C$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  end}
```

Prior work: Syntactic duality

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

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 _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_C$   $A$   $B$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_A$   $B$   $C$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  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:

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

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 _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_C$   $A$   $B$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_A$   $B$   $C$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  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

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

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 _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_C$   $A$   $B$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_A$   $B$   $C$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  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

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

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

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

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

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

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

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

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

$$\frac{\text{CHAN-SUB} \quad c \multimap p_1 \quad p_1 \sqsubseteq p_2}{c \multimap p_2}^*$$

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

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

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

$$\frac{\text{CHAN-SUB} \quad c \multimap p_1 \quad p_1 \sqsubseteq p_2}{c \multimap p_2} *$$

$$\frac{\text{WP-TRY-SEND} \quad c \multimap q \quad q \sqsubseteq ![i] \{P\}.p \quad P}{\text{wp } c[i].\mathbf{try_send}() \{ b. \mathbf{if } b \mathbf{ then } c \multimap p \mathbf{ else } c \multimap q * P \}} *$$

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

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

$$\frac{\text{CHAN-SUB} \quad c \multimap p_1 \quad p_1 \sqsubseteq p_2}{c \multimap p_2} *$$

$$\frac{\text{WP-TRY-SEND} \quad c \multimap q \quad q \sqsubseteq ![i] \{P\}.p \quad P}{\text{wp } c[i].\mathbf{try_send}() \{ b. \mathbf{if } b \mathbf{ then } c \multimap p \mathbf{ else } c \multimap q * P \}} *$$

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

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

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 _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_C$   $A$   $B$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_A$   $B$   $C$  () with none  $\Rightarrow$  (); some _  $\Rightarrow$  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  $\_ \Rightarrow \text{free } \ell$  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}
```

Protocols:

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

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 _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_C$  A B () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  end};  
  fork {match send_recv  $c_A$  B C () with none  $\Rightarrow$  (); some _  $\Rightarrow$  free  $\ell$  end}
```

Protocols:

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

Verified functional correctness!

Concluding Remarks

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

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

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] ⟨“Thanks”⟩ {MixtrisOverview}.
μrec. (?[A] (q : Question i) ⟨q⟩ {AboutMultris q}.
    ! [A] (a : Answer) ⟨a⟩ {Insightful q a}. rec)
+
! [C] ⟨“Times up?”⟩. end

```

Backup Slides

Chang and Roberts's Protocol

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

Chang and Roberts's Protocol

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

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

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

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

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

Implementation - Synchronisation Cells

`new_sync () := ref none`

`wait c := match !c with
| none $\Rightarrow ()$
| some _ \Rightarrow wait c
end`

`sync_put c v := c \leftarrow some v;
 wait c.`

`sync_get c :=
 match Xchg c none with
 | none \Rightarrow sync_get c
 | some v \Rightarrow v
 end`

`sync_try_put c v :=
 c \leftarrow some v;
 match Xchg c none with
 | none \Rightarrow true
 | some _ \Rightarrow 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** $((m, 0), \dots, (m, n - 1))$

$c[j].\mathbf{send}(v)$:= **let** $(m, i) := c$ **in**
 sync_put $m_{i,j}$ v

$c[j].\mathbf{recv}()$:= **let** $(m, i) := c$ **in**
 sync_get $m_{j,i}$

$c[j].\mathbf{try_send}(v)$:= **let** $(m, i) := c$ **in**
 sync_try_put $m_{i,j}$ v

$c[j].\mathbf{try_recv}()$:= **let** $(m, i) := c$ **in**
 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

$$\frac{\text{CONSISTENT } \vec{p}}{\vdash \exists \chi. \text{prot_ctx } \chi \mid \vec{p} \mid * \bigstar_{i \mapsto p \in \vec{p}} \text{prot_own } \chi \ i \ p}^*$$

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

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

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