Dependent Multiparty Communication in Separation Logic

Jonas Kastberg Hinrichsen
Aarhus University

Jules Jacobs
Cornell University

Robbert Krebbers
Radboud University Nijmegen

Multiparty Message Passing Concurrency

Multiparty message passing concurrency:

- Well-structured approach to writing concurrent programs
- Communication between pairs depend on communication between others

Multiparty Message Passing Concurrency

Multiparty message passing concurrency:

- Well-structured approach to writing concurrent programs
- Communication between pairs depend on communication between others

Many use cases exist:

- Consensus algorithms: Leader election
- Map-reduce
- Multiparty computation

Multiparty Message Passing Concurrency

Multiparty message passing concurrency:

- Well-structured approach to writing concurrent programs
- Communication between pairs depend on communication between others

Many use cases exist:

- Consensus algorithms: Leader election
- Map-reduce
- Multiparty computation

Why we care:

- Communication is a great (and necessary) abstraction barrier
- Many errors happen between abstractions

Multiparty Message Passing Concurrency

Multiparty message passing concurrency:

- Well-structured approach to writing concurrent programs
- Communication between pairs depend on communication between others

Many use cases exist:

- Consensus algorithms: Leader election
- Map-reduce
- Multiparty computation

Why we care:

- Communication is a great (and necessary) abstraction barrier
- Many errors happen between abstractions

We consider:

- Synchronous multiparty communication: Actors block until synchronisation
- Shared-memory concurrency: ML-like language

Multiparty Message Passing Concurrency in Shared Memory

Multiparty channels in shared memory:

new_chan n	Create a network of n endpoints with channels between all pairs
c.**send**$[i](v)$	Send value v from endpoint c to participant i
c.**recv**$[i]$	Receive next value on endpoint c from participant i

Multiparty Message Passing Concurrency in Shared Memory

Multiparty channels in shared memory:

$\mathbf{new\_chan}\ n$	Create a network of n endpoints with channels between all pairs
$c.\mathbf{send}[i](v)$	Send value v from endpoint c to participant i
$c.\mathbf{recv}[i]$	Receive next value on endpoint c from participant i

Example Program: Roundtrip

$$
\begin{array}{l}
\mathbf{let}\ (c_0, c_1, c_2) = \mathbf{new\_chan}\ 3\ \mathbf{in} \\
\mathbf{fork}\ \{\mathbf{let}\ x = c_1.\mathbf{recv}[0]\ \mathbf{in}\ c_1.\mathbf{send}[2](x+1)\}\ ; \\
\mathbf{fork}\ \{\mathbf{let}\ x = c_2.\mathbf{recv}[1]\ \mathbf{in}\ c_2.\mathbf{send}[0](x+1)\}\ ; \\
c_0.\mathbf{send}[1](40);\ \mathbf{let}\ x = c_0.\mathbf{recv}[2]\ \mathbf{in}\ \mathbf{assert}(x = 42)
\end{array}
$$

Safety and Functional Correctness

Example Program: Roundtrip

$$
\begin{array}{l}
\mathtt{let}\,(c_0, c_1, c_2) = \mathtt{new\_chan}\; 3\; \mathtt{in} \\
\mathtt{fork}\;\{\mathtt{let}\, x = c_1.\mathtt{recv}[0]\;\mathtt{in}\; c_1.\mathtt{send}[2](x+1)\}\,; \\
\mathtt{fork}\;\{\mathtt{let}\, x = c_2.\mathtt{recv}[1]\;\mathtt{in}\; c_2.\mathtt{send}[0](x+1)\}\,; \\
c_0.\mathtt{send}[1](40);\, \mathtt{let}\, x = c_0.\mathtt{recv}[2]\;\mathtt{in}\;\mathtt{assert}(x = 42)
\end{array}
$$

Goal: Prove crash-freedom (safety) and verify asserts (functional correctness)

Safety and Functional Correctness

Example Program: Roundtrip

$$\begin{array}{l}
\mathtt{let}\,(c_0, c_1, c_2) = \mathtt{new_chan}\,3\,\mathtt{in} \\
\mathtt{fork}\ \{\mathtt{let}\,x = c_1.\mathtt{recv}[0]\,\mathtt{in}\,c_1.\mathtt{send}[2](x+1)\}\,; \\
\mathtt{fork}\ \{\mathtt{let}\,x = c_2.\mathtt{recv}[1]\,\mathtt{in}\,c_2.\mathtt{send}[0](x+1)\}\,; \\
c_0.\mathtt{send}[1](40);\,\mathtt{let}\,x = c_0.\mathtt{recv}[2]\,\mathtt{in}\ \mathtt{assert}(x = 42)
\end{array}$$

Goal: Prove crash-freedom (safety) and verify asserts (functional correctness)

Safety	Functional Correctness
Type systems	Program logics

Safety and Functional Correctness

Example Program: Roundtrip

$$\begin{array}{l}
\mathtt{let}\,(c_0, c_1, c_2) = \mathtt{new_chan}\;3\;\mathtt{in} \\
\mathtt{fork}\;\{\mathtt{let}\,x = c_1.\mathtt{recv}[0]\;\mathtt{in}\;c_1.\mathtt{send}[2](x+1)\}\,; \\
\mathtt{fork}\;\{\mathtt{let}\,x = c_2.\mathtt{recv}[1]\;\mathtt{in}\;c_2.\mathtt{send}[0](x+1)\}\,; \\
c_0.\mathtt{send}[1](40);\,\mathtt{let}\,x = c_0.\mathtt{recv}[2]\;\mathtt{in}\;\mathtt{assert}(x = 42)
\end{array}$$

Goal: Prove crash-freedom (safety) and verify asserts (functional correctness)

Safety	**Functional Correctness**
Type systems	Program logics
Automatic checking	Manual proofs

Safety and Functional Correctness

Example Program: Roundtrip

$$\begin{array}{l}\texttt{let}\,(c_0, c_1, c_2) = \texttt{new\_chan}\;3\;\texttt{in} \\ \texttt{fork}\;\{\texttt{let}\,x = c_1.\texttt{recv}[0]\;\texttt{in}\;c_1.\texttt{send}[2](x+1)\}\,; \\ \texttt{fork}\;\{\texttt{let}\,x = c_2.\texttt{recv}[1]\;\texttt{in}\;c_2.\texttt{send}[0](x+1)\}\,; \\ c_0.\texttt{send}[1](40);\,\texttt{let}\,x = c_0.\texttt{recv}[2]\;\texttt{in}\;\texttt{assert}(x = 42)\end{array}$$

Goal: Prove crash-freedom (safety) and verify asserts (functional correctness)

Safety	Functional Correctness
Type systems	Program logics
Automatic checking	Manual proofs
Multiparty session types: $\mathbf{!}[1]\mathbb{Z}.\,\mathbf{?}[2]\mathbb{Z}.\,\mathbf{end}$	???

! is send, **?** is receive

Key Idea

Prior Work: Binary dependent separation protocols (DSP's)

- **Safety:** $!\mathbb{Z}.\ ?\mathbb{Z}.\ \textbf{end}$
- **Functional Correctness:** $!\,(x : \mathbb{Z})\,\langle x\rangle.\ ?\langle x + 2\rangle.\ \textbf{end}$

Key Idea

Prior Work: Binary dependent separation protocols (DSP's)

- **Safety:** $\mathbf{!}\mathbb{Z}.\,\mathbf{?}\mathbb{Z}.\,\mathbf{end}$
- **Functional Correctness:** $\mathbf{!}\,(x : \mathbb{Z})\,\langle x \rangle.\,\mathbf{?}\langle x + 2 \rangle.\,\mathbf{end}$

Key Idea: Multiparty dependent separation protocols!

- **Safety:** $\mathbf{!}[i]\mathbb{Z}.\,\mathbf{?}[j]\mathbb{Z}.\,\mathbf{end}$
- **Functional Correctness:** $\mathbf{!}\,[i]\,(x : \mathbb{Z})\,\langle x \rangle.\,\mathbf{?}[j]\,\langle x + 2 \rangle.\,\mathbf{end}$

Key Idea

Prior Work: Binary dependent separation protocols (DSP's)

- **Safety:** $!\mathbb{Z}.\,?\mathbb{Z}.\,\textsf{end}$
- **Functional Correctness:** $!\,(x : \mathbb{Z})\,\langle x \rangle.\,?\langle x + 2 \rangle.\,\textsf{end}$

Key Idea: Multiparty dependent separation protocols!

- **Safety:** $![i]\mathbb{Z}.\,?[j]\mathbb{Z}.\,\textsf{end}$
- **Functional Correctness:** $!\,[i]\,(x : \mathbb{Z})\,\langle x \rangle.\,?[j]\,\langle x + 2 \rangle.\,\textsf{end}$

Example Program: Roundtrip

$$c_0.\mathtt{send}[1](40);\, \mathtt{let}\, x = c_0.\mathtt{recv}[2]\; \mathtt{in}\; \mathtt{assert}(x = 42)$$

Key Idea

Prior Work: Binary dependent separation protocols (DSP's)

- **Safety:** $!\mathbb{Z}.\,?\mathbb{Z}.\,\textbf{end}$
- **Functional Correctness:** $!\,(x:\mathbb{Z})\,\langle x\rangle.\,?\langle x+2\rangle.\,\textbf{end}$

Key Idea: Multiparty dependent separation protocols!

- **Safety:** $![i]\mathbb{Z}.\,?[j]\mathbb{Z}.\,\textbf{end}$
- **Functional Correctness:** $!\,[i]\,(x:\mathbb{Z})\,\langle x\rangle.\,?[j]\,\langle x+2\rangle.\,\textbf{end}$

Example Program: Roundtrip

$$c_0.\mathtt{send}[1](40);\,\mathtt{let}\,x = c_0.\mathtt{recv}[2]\;\mathtt{in}\;\mathtt{assert}(x = 42)$$

Challenge: How to guarantee consistent global communication?

Challenge

Challenge: How to guarantee consistent global communication?

$$\begin{array}{l}
\mathbf{let}\,(c_0, c_1, c_2) = \mathbf{new_chan}\ 3\ \mathbf{in} \\
\mathbf{fork}\ \{\mathbf{let}\, x = c_1.\mathbf{recv}[0]\ \mathbf{in}\, c_1.\mathbf{send}[2](x+1)\}\,; \\
\mathbf{fork}\ \{\mathbf{let}\, x = c_2.\mathbf{recv}[1]\ \mathbf{in}\, c_2.\mathbf{send}[0](x+1)\}\,; \\
c_0.\mathbf{send}[1](40);\, \mathbf{let}\, x = c_0.\mathbf{recv}[2]\ \mathbf{in}\ \mathbf{assert}(x = 42)
\end{array}$$

Challenge

Challenge: How to guarantee consistent global communication?

$$\begin{array}{l}
\mathtt{let}\,(c_0, c_1, c_2) = \mathtt{new_chan}\,3\,\mathtt{in} \\
\mathtt{fork}\ \{\mathtt{let}\,x = c_1.\mathtt{recv}[0]\,\mathtt{in}\,c_1.\mathtt{send}[2](x+1)\}\,; \\
\mathtt{fork}\ \{\mathtt{let}\,x = c_2.\mathtt{recv}[1]\,\mathtt{in}\,c_2.\mathtt{send}[0](x+1)\}\,; \\
c_0.\mathtt{send}[1](40);\,\mathtt{let}\,x = c_0.\mathtt{recv}[2]\,\mathtt{in}\,\mathtt{assert}(x = 42)
\end{array}$$

Prior work: Syntactic duality

$$\begin{array}{lcl}
c_0 & : & \mathbf{!}[1]\mathbb{Z}.\,\mathbf{?}[2]\mathbb{Z}.\,\mathbf{end} \\
c_1 & : & \mathbf{?}[0]\mathbb{Z}.\,\mathbf{!}[2]\mathbb{Z}.\,\mathbf{end} \\
c_2 & : & \mathbf{?}[1]\mathbb{Z}.\,\mathbf{!}[0]\mathbb{Z}.\,\mathbf{end}
\end{array}$$

Challenge: How to guarantee consistent global communication?

$$\begin{array}{l}
\mathtt{let}\,(c_0, c_1, c_2) = \mathtt{new_chan}\;3\;\mathtt{in} \\
\mathtt{fork}\;\{\mathtt{let}\,x = c_1.\mathtt{recv}[0]\;\mathtt{in}\;c_1.\mathtt{send}[2](x+1)\}\,; \\
\mathtt{fork}\;\{\mathtt{let}\,x = c_2.\mathtt{recv}[1]\;\mathtt{in}\;c_2.\mathtt{send}[0](x+1)\}\,; \\
c_0.\mathtt{send}[1](40);\mathtt{let}\,x = c_0.\mathtt{recv}[2]\;\mathtt{in}\;\mathtt{assert}(x = 42)
\end{array}$$

Prior work: Syntactic duality

$$\begin{array}{lcl}
c_0 & : & \mathbf{!}[1]\mathbb{Z}.\,\mathbf{?}[2]\mathbb{Z}.\,\mathbf{end} \\
c_1 & : & \mathbf{?}[0]\mathbb{Z}.\,\mathbf{!}[2]\mathbb{Z}.\,\mathbf{end} \\
c_2 & : & \mathbf{?}[1]\mathbb{Z}.\,\mathbf{!}[0]\mathbb{Z}.\,\mathbf{end}
\end{array}$$

This work:

$$c_0 \rightarrowtail \mathbf{!}\,[1]\,(x:\mathbb{Z})\,\langle x\rangle.\,\mathbf{?}[2]\,\langle x+2\rangle.\,\mathbf{end}$$

Challenge

Challenge: How to guarantee consistent global communication?

$$
\begin{array}{l}
\mathbf{let}\,(c_0, c_1, c_2) = \mathbf{new\_chan}\ 3\ \mathbf{in} \\
\mathbf{fork}\ \{\mathbf{let}\ x = c_1.\mathbf{recv}[0]\ \mathbf{in}\ c_1.\mathbf{send}[2](x+1)\}\,; \\
\mathbf{fork}\ \{\mathbf{let}\ x = c_2.\mathbf{recv}[1]\ \mathbf{in}\ c_2.\mathbf{send}[0](x+1)\}\,; \\
c_0.\mathbf{send}[1](40);\mathbf{let}\ x = c_0.\mathbf{recv}[2]\ \mathbf{in}\ \mathbf{assert}(x = 42)
\end{array}
$$

Prior work: Syntactic duality

$$
\begin{array}{lcl}
c_0 & : & \mathbf{!}[1]\mathbb{Z}.\,\mathbf{?}[2]\mathbb{Z}.\,\mathbf{end} \\
c_1 & : & \mathbf{?}[0]\mathbb{Z}.\,\mathbf{!}[2]\mathbb{Z}.\,\mathbf{end} \\
c_2 & : & \mathbf{?}[1]\mathbb{Z}.\,\mathbf{!}[0]\mathbb{Z}.\,\mathbf{end}
\end{array}
$$

This work:

$$
\begin{array}{l}
c_0 \rightarrowtail \mathbf{!}\,[1]\,(x : \mathbb{Z})\,\langle x \rangle.\,\mathbf{?}[2]\,\langle x + 2 \rangle.\,\mathbf{end} \\
c_1 \rightarrowtail \mathbf{?}[0]\,(x : \mathbb{Z})\,\langle x \rangle.\,\mathbf{!}\,[2]\,\langle x + 1 \rangle.\,\mathbf{end}
\end{array}
$$

Challenge

Challenge: How to guarantee consistent global communication?

$$
\begin{array}{l}
\mathtt{let}\,(c_0, c_1, c_2) = \mathtt{new\_chan}\,3\,\mathtt{in} \\
\mathtt{fork}\ \{\mathtt{let}\,x = c_1.\mathtt{recv}[0]\,\mathtt{in}\,c_1.\mathtt{send}[2](x+1)\}\,; \\
\mathtt{fork}\ \{\mathtt{let}\,x = c_2.\mathtt{recv}[1]\,\mathtt{in}\,c_2.\mathtt{send}[0](x+1)\}\,; \\
c_0.\mathtt{send}[1](40);\,\mathtt{let}\,x = c_0.\mathtt{recv}[2]\,\mathtt{in}\,\mathtt{assert}(x = 42)
\end{array}
$$

Prior work: Syntactic duality

$$
\begin{array}{lcl}
c_0 & : & ![1]\mathbb{Z}.\,?[2]\mathbb{Z}.\,\mathbf{end} \\
c_1 & : & ?[0]\mathbb{Z}.\,![2]\mathbb{Z}.\,\mathbf{end} \\
c_2 & : & ?[1]\mathbb{Z}.\,![0]\mathbb{Z}.\,\mathbf{end}
\end{array}
$$

This work:

$$
\begin{array}{l}
c_0 \rightarrowtail\, !\,[1]\,(x : \mathbb{Z})\,\langle x \rangle.\,?[2]\,\langle x+2 \rangle.\,\mathbf{end} \\
c_1 \rightarrowtail\, ?[0]\,(x : \mathbb{Z})\,\langle x \rangle.\,!\,[2]\,\langle x+1 \rangle.\,\mathbf{end} \\
c_2 \rightarrowtail\, ?[1]\,(x : \mathbb{Z})\,\langle x \rangle.\,!\,[0]\,\langle x+1 \rangle.\,\mathbf{end}
\end{array}
$$

Challenge

Challenge: How to guarantee consistent global communication?

$$
\begin{array}{l}
\mathtt{let}\,(c_0, c_1, c_2) = \mathtt{new\_chan}\,3\,\mathtt{in} \\
\mathtt{fork}\ \{\mathtt{let}\,x = c_1.\mathtt{recv}[0]\,\mathtt{in}\,c_1.\mathtt{send}[2](x+1)\}\,; \\
\mathtt{fork}\ \{\mathtt{let}\,x = c_2.\mathtt{recv}[1]\,\mathtt{in}\,c_2.\mathtt{send}[0](x+1)\}\,; \\
c_0.\mathtt{send}[1](40);\,\mathtt{let}\,x = c_0.\mathtt{recv}[2]\,\mathtt{in}\,\mathtt{assert}(x = 42)
\end{array}
$$

Prior work: Syntactic duality

$$
\begin{array}{lcl}
c_0 & : & \mathbf{!}[1]\mathbb{Z}.\,\mathbf{?}[2]\mathbb{Z}.\,\mathbf{end} \\
c_1 & : & \mathbf{?}[0]\mathbb{Z}.\,\mathbf{!}[2]\mathbb{Z}.\,\mathbf{end} \\
c_2 & : & \mathbf{?}[1]\mathbb{Z}.\,\mathbf{!}[0]\mathbb{Z}.\,\mathbf{end}
\end{array}
$$

This work: Semantic duality

$$
\begin{array}{l}
c_0 \rightarrowtail \mathbf{!}\,[1]\,(x:\mathbb{Z})\,\langle x\rangle.\,\mathbf{?}[2]\,\langle x+2\rangle.\,\mathbf{end} \\
c_1 \rightarrowtail \mathbf{?}[0]\,(x:\mathbb{Z})\,\langle x\rangle.\,\mathbf{!}\,[2]\,\langle x+1\rangle.\,\mathbf{end} \\
c_2 \rightarrowtail \mathbf{?}[1]\,(x:\mathbb{Z})\,\langle x\rangle.\,\mathbf{!}\,[0]\,\langle x+1\rangle.\,\mathbf{end}
\end{array}
$$

Challenge

Challenge: How to guarantee consistent global communication?

$$\begin{array}{l}
\mathtt{let}\,(c_0, c_1, c_2) = \mathtt{new_chan}\ 3\ \mathtt{in} \\
\mathtt{fork}\ \{\mathtt{let}\ x = c_1.\mathtt{recv}[0]\ \mathtt{in}\ c_1.\mathtt{send}[2](x+1)\}\,; \\
\mathtt{fork}\ \{\mathtt{let}\ x = c_2.\mathtt{recv}[1]\ \mathtt{in}\ c_2.\mathtt{send}[0](x+1)\}\,; \\
c_0.\mathtt{send}[1](40);\ \mathtt{let}\ x = c_0.\mathtt{recv}[2]\ \mathtt{in}\ \mathtt{assert}(x = 42)
\end{array}$$

Prior work: Syntactic duality

$$\begin{array}{lcl}
c_0 & : & ![1]\mathbb{Z}.\,?[2]\mathbb{Z}.\,\mathbf{end} \\
c_1 & : & ?[0]\mathbb{Z}.\,![2]\mathbb{Z}.\,\mathbf{end} \\
c_2 & : & ?[1]\mathbb{Z}.\,![0]\mathbb{Z}.\,\mathbf{end}
\end{array}$$

This work: Semantic duality

$$\begin{array}{l}
c_0 \rightarrowtail\, ![1]\,(x:\mathbb{Z})\,\langle x\rangle.\,?[2]\,\langle x+2\rangle.\,\mathbf{end} \\
c_1 \rightarrowtail\, ?[0]\,(x:\mathbb{Z})\,\langle x\rangle.\,![2]\,\langle x+1\rangle.\,\mathbf{end} \\
c_2 \rightarrowtail\, ?[1]\,(x:\mathbb{Z})\,\langle x\rangle.\,![0]\,\langle x+1\rangle.\,\mathbf{end}
\end{array}$$

Key Idea: Define and prove consistency via separation logic!

Contributions

Multiparty dependent separation protocols (MDSPs)

- Rich specification language for describing multiparty communication
- Protocol consistency defined in terms of semantic duality

Multiparty Actris

- Program logic for multiparty communication via MDSPs in Iris
- Support for language-parametric instantiation of Multiparty Actris

Verification of suite of multiparty programs

- Increasingly intricate variations of the roundtrip program
- Chang and Roberts ring leader election algorithm

Full mechanisation in Coq

- With tactic support

Roadmap of this talk

Tour of Multiparty Actris

- Multiparty dependent separation protocols and protocol consistency
- Program logic rules
- Verification of suite of roundtrip variations

Verification of Chang and Roberts ring leader election algorithm

- Overview of algorithm
- Ring leader election protocol
- Verification of algorithm

Language-parametricity of Multiparty Actris

- Multiparty Actris ghost theory

Conclusion and Future Work

Tour of Multiparty Actris

Roundtrip Example

Roundtrip program:

$$
\begin{array}{l}
\mathtt{let}\,(c_0, c_1, c_2) = \mathtt{new\_chan}\;3\;\mathtt{in} \\
\mathtt{fork}\;\{\mathtt{let}\,x = c_1.\mathtt{recv}[0]\;\mathtt{in}\;c_1.\mathtt{send}[2](x+1)\}\,; \\
\mathtt{fork}\;\{\mathtt{let}\,x = c_2.\mathtt{recv}[1]\;\mathtt{in}\;c_2.\mathtt{send}[0](x+1)\}\,; \\
c_0.\mathtt{send}[1](40); \mathtt{let}\,x = c_0.\mathtt{recv}[2]\;\mathtt{in}\;\mathtt{assert}(x = 42)
\end{array}
$$

Protocols:

$$
\begin{array}{l}
c_0 \rightarrowtail \,!\,[1]\,(x : \mathbb{Z})\,\langle x \rangle.\,?[2]\,\langle x+2 \rangle.\,\mathbf{end} \\
c_1 \rightarrowtail \,?[0]\,(x : \mathbb{Z})\,\langle x \rangle.\,!\,[2]\,\langle x+1 \rangle.\,\mathbf{end} \\
c_2 \rightarrowtail \,?[1]\,(x : \mathbb{Z})\,\langle x \rangle.\,!\,[0]\,\langle x+1 \rangle.\,\mathbf{end}
\end{array}
$$

Syntax:

$$t, u, P, Q, p ::= \ \ldots \mid \mathbf{!}\,[i]\,\vec{x}:\vec{\tau}\,\langle v\rangle.\,p \mid \mathbf{?}[i]\,\vec{x}:\vec{\tau}\,\langle v\rangle.\,p \mid \textbf{end} \mid c \rightarrowtail p \mid \ldots$$

Rules:

HT-SEND

$$\{c \rightarrowtail \mathbf{!}\,[i]\,\vec{x}:\vec{\tau}\,\langle v\rangle.\,p\}\; c.\textbf{send}[i](v[\vec{t}/\vec{x}])\; \{c \rightarrowtail p[\vec{t}/\vec{x}]\}$$

HT-RECV

$$\{c \rightarrowtail \mathbf{?}[i]\,\vec{x}:\vec{\tau}\,\langle v\rangle.\,p\}\; c.\mathbf{recv}[i]\; \{w.\, \exists \vec{y}.\, w = v[\vec{y}/\vec{x}] * c \rightarrowtail p[\vec{y}/\vec{x}]\}$$

HT-NEW

$$\{\texttt{prot\_consistent}\; ps\}$$

$$\mathbf{new\_chan}\; |ps|$$

$$\{(c_0, \ldots, c_{(|ps|-1)}).\, c_0 \rightarrowtail ps[0] * \ldots * c_{(|ps|-1)} \rightarrowtail ps[|ps|-1]\}$$

Protocol Consistency

For any synchronised exchange from i to j, the binders of i must be sufficient to:

1. Instantiate the binders of j
2. Prove equality of exchanged values
3. Prove protocol consistency where i and j are updated to their respective tails

Repeat until no more synchronised exchanges exist.

Protocol Consistency

For any synchronised exchange from i to j, the binders of i must be sufficient to:

1. Instantiate the binders of j
2. Prove equality of exchanged values
3. Prove protocol consistency where i and j are updated to their respective tails

Repeat until no more synchronised exchanges exist.

Example:

$$\begin{aligned} ps[0] &:= \,!\,[1]\,(x : \mathbb{Z})\,\langle x\rangle.\,?[2]\,\langle x+2\rangle.\,\mathbf{end} \\ ps[1] &:= \,?[0]\,(x : \mathbb{Z})\,\langle x\rangle.\,!\,[2]\,\langle x+1\rangle.\,\mathbf{end} \\ ps[2] &:= \,?[1]\,(x : \mathbb{Z})\,\langle x\rangle.\,!\,[0]\,\langle x+1\rangle.\,\mathbf{end} \end{aligned}$$

Protocol Consistency

For any synchronised exchange from i to j, the binders of i must be sufficient to:

1. Instantiate the binders of j
2. Prove equality of exchanged values
3. Prove protocol consistency where i and j are updated to their respective tails

Repeat until no more synchronised exchanges exist.

Example:

$$
\begin{aligned}
ps[0] &:= \,!\,[1]\,(x : \mathbb{Z})\,\langle x \rangle.\,?[2]\,\langle x + 2 \rangle.\,\mathbf{end} \\
ps[1] &:= \,?[0]\,(x : \mathbb{Z})\,\langle x \rangle.\,!\,[2]\,\langle x + 1 \rangle.\,\mathbf{end} \\
ps[2] &:= \,?[1]\,(x : \mathbb{Z})\,\langle x \rangle.\,!\,[0]\,\langle x + 1 \rangle.\,\mathbf{end}
\end{aligned}
$$

Note that:

- Non-determinism may occur, resulting in tree of subgoals
- Protocols that do not synchronise are always valid
  - This is safe, as the corresponding program would diverge

Roundtrip Example - Verified

Roundtrip program:

$$
\begin{array}{l}
\mathbf{let}\,(c_0, c_1, c_2) = \mathbf{new\_chan}\;3\;\mathbf{in} \\
\mathbf{fork}\;\{\mathbf{let}\,x = c_1.\mathbf{recv}[0]\;\mathbf{in}\;c_1.\mathbf{send}[2](x+1)\}\,; \\
\mathbf{fork}\;\{\mathbf{let}\,x = c_2.\mathbf{recv}[1]\;\mathbf{in}\;c_2.\mathbf{send}[0](x+1)\}\,; \\
c_0.\mathbf{send}[1](40);\,\mathbf{let}\,x = c_0.\mathbf{recv}[2]\;\mathbf{in}\;\mathbf{assert}(x = 42)
\end{array}
$$

Protocols:

$$
\begin{array}{l}
c_0 \rightarrowtail \,!\,[1]\,(x : \mathbb{Z})\,\langle x \rangle.\,?[2]\,\langle x+2 \rangle.\,\mathbf{end} \\
c_1 \rightarrowtail \,?[0]\,(x : \mathbb{Z})\,\langle x \rangle.\,!\,[2]\,\langle x+1 \rangle.\,\mathbf{end} \\
c_2 \rightarrowtail \,?[1]\,(x : \mathbb{Z})\,\langle x \rangle.\,!\,[0]\,\langle x+1 \rangle.\,\mathbf{end}
\end{array}
$$

Verified Safety!

Roundtrip Reference Example

Roundtrip reference program:

$$
\begin{array}{l}
\mathtt{let}\,(c_0, c_1, c_2) = \mathtt{new\_chan}\;3\;\mathtt{in} \\
\mathtt{fork}\;\{\mathtt{let}\,\ell = c_1.\mathtt{recv}[0]\;\mathtt{in}\;\ell \leftarrow (!\,\ell + 1); c_1.\mathtt{send}[2](\ell)\}\,; \\
\mathtt{fork}\;\{\mathtt{let}\,\ell = c_2.\mathtt{recv}[1]\;\mathtt{in}\;\ell \leftarrow (!\,\ell + 1); c_2.\mathtt{send}[0]()\}\,; \\
\mathtt{let}\,\ell = \mathtt{ref}\,40\;\mathtt{in}\;c_0.\mathtt{send}[1](\ell); c_0.\mathtt{recv}[2]; \mathtt{let}\,x = !\,\ell\;\mathtt{in}\;\mathtt{assert}(x = 42)
\end{array}
$$

Roundtrip Reference Example

Roundtrip reference program:

$$
\begin{array}{l}
\mathtt{let}\,(c_0, c_1, c_2) = \mathtt{new\_chan}\,3\;\mathtt{in} \\
\mathtt{fork}\;\{\mathtt{let}\,\ell = c_1.\mathtt{recv}[0]\;\mathtt{in}\;\ell \leftarrow (!\,\ell + 1); c_1.\mathtt{send}[2](\ell)\}\,; \\
\mathtt{fork}\;\{\mathtt{let}\,\ell = c_2.\mathtt{recv}[1]\;\mathtt{in}\;\ell \leftarrow (!\,\ell + 1); c_2.\mathtt{send}[0](())\}\,; \\
\mathtt{let}\,\ell = \mathtt{ref}\,40\;\mathtt{in}\;c_0.\mathtt{send}[1](\ell); c_0.\mathtt{recv}[2]; \mathtt{let}\,x = !\,\ell\;\mathtt{in}\;\mathtt{assert}(x = 42)
\end{array}
$$

Protocols:

$$
\begin{array}{l}
c_0 \rightarrowtail \mathbf{!}\,[1]\,(\ell : \mathsf{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{?}[2]\,\langle () \rangle \{\ell \mapsto (x+2)\}.\,\mathbf{end} \\
c_1 \rightarrowtail \mathbf{?}[0]\,(\ell : \mathsf{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,[2]\,\langle \ell \rangle \{\ell \mapsto (x+1)\}.\,\mathbf{end} \\
c_2 \rightarrowtail \mathbf{?}[1]\,(\ell : \mathsf{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,[0]\,\langle () \rangle \{\ell \mapsto (x+1)\}.\,\mathbf{end}
\end{array}
$$

Multiparty Actris with Resources

Syntax:

$$t, u, P, Q, p ::= \ \ldots \mid \mathbf{!}\,[i]\,\vec{x}:\vec{\tau}\langle v\rangle\{P\}.\,p \mid \mathbf{?}\,[i]\,\vec{x}:\vec{\tau}\langle v\rangle\{P\}.\,p \mid$$

Rules:

Ht-send

$$\{c \rightarrowtail \mathbf{!}\,[i]\,\vec{x}:\vec{\tau}\langle v\rangle\{P\}.\,p * P[\vec{t}/\vec{x}]\}\ c.\mathbf{send}[i](v[\vec{t}/\vec{x}])\ \{c \rightarrowtail p[\vec{t}/\vec{x}]\}$$

Ht-recv

$$\{c \rightarrowtail \mathbf{?}\,[i]\,\vec{x}:\vec{\tau}\langle v\rangle\{P\}.\,p\}\ c.\mathbf{recv}[i]\ \{w.\ \exists \vec{y}.\ w = v[\vec{y}/\vec{x}] * c \rightarrowtail p[\vec{y}/\vec{x}] * P[\vec{y}/\vec{x}]\}$$

Ht-new

$$\{\texttt{prot\_consistent}\ ps\}$$

$$\mathbf{new\_chan}\ |ps|$$

$$\{(c_0, \ldots, c_{(|ps|-1)}).\ c_0 \rightarrowtail ps[0] * \ldots * c_{(|ps|-1)} \rightarrowtail ps[|ps|-1]\}$$

Protocol Consistency with Resources

For any synchronised exchange from i to j, the binders and resources of i must be sufficient to:

1. Instantiate the binders of j
2. Prove equality of exchanged values and the resources of j
3. Prove protocol consistency where i and j are updated to their respective tails

Repeat until no more synchronised exchanges exist.

Example:

$$
\begin{aligned}
ps[0] &:= \mathbf{!}\,[1]\,(\ell : \mathsf{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{?}[2]\,\langle () \rangle \{\ell \mapsto (x+2)\}.\,\mathbf{end} \\
ps[1] &:= \mathbf{?}[0]\,(\ell : \mathsf{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,[2]\,\langle \ell \rangle \{\ell \mapsto (x+1)\}.\,\mathbf{end} \\
ps[2] &:= \mathbf{?}[1]\,(\ell : \mathsf{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,[0]\,\langle () \rangle \{\ell \mapsto (x+1)\}.\,\mathbf{end}
\end{aligned}
$$

Roundtrip Reference Example - Verified

Roundtrip reference program:

$$
\begin{aligned}
&\mathtt{let}\,(c_0, c_1, c_2) = \mathtt{new\_chan}\;3\;\mathtt{in}\\
&\mathtt{fork}\;\{\mathtt{let}\,\ell = c_1.\mathtt{recv}[0]\;\mathtt{in}\;\ell \leftarrow (!\,\ell + 1); c_1.\mathtt{send}[2](\ell)\};\\
&\mathtt{fork}\;\{\mathtt{let}\,\ell = c_2.\mathtt{recv}[1]\;\mathtt{in}\;\ell \leftarrow (!\,\ell + 1); c_2.\mathtt{send}[0](())\};\\
&\mathtt{let}\,\ell = \mathtt{ref}\,40\;\mathtt{in}\;c_0.\mathtt{send}[1](\ell); c_0.\mathtt{recv}[2]; \mathtt{let}\,x = !\,\ell\;\mathtt{in}\;\mathtt{assert}(x = 42)
\end{aligned}
$$

Protocols:

$$
\begin{aligned}
c_0 &\rightarrowtail \mathbf{!}\,[1]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle\ell\rangle\{\ell \mapsto x\}.\,\mathbf{?}[2]\,\langle()\rangle\{\ell \mapsto (x+2)\}.\,\mathbf{end}\\
c_1 &\rightarrowtail \mathbf{?}[0]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle\ell\rangle\{\ell \mapsto x\}.\,\mathbf{!}\,[2]\,\langle\ell\rangle\{\ell \mapsto (x+1)\}.\,\mathbf{end}\\
c_2 &\rightarrowtail \mathbf{?}[1]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle\ell\rangle\{\ell \mapsto x\}.\,\mathbf{!}\,[0]\,\langle()\rangle\{\ell \mapsto (x+1)\}.\,\mathbf{end}
\end{aligned}
$$

Roundtrip Reference Recursion Example

Roundtrip reference recursion program:

$$
\begin{aligned}
&\mathtt{let}\,(c_0, c_1, c_2) = \mathtt{new\_chan}\,3\,\mathtt{in}\\
&\mathtt{fork}\ \{\mathtt{loop}(\mathtt{let}\,\ell = c_1.\mathtt{recv}[0]\,\mathtt{in}\,\ell \leftarrow (!\,\ell + 1); c_1.\mathtt{send}[2](\ell))\}\,;\\
&\mathtt{fork}\ \{\mathtt{loop}(\mathtt{let}\,\ell = c_2.\mathtt{recv}[1]\,\mathtt{in}\,\ell \leftarrow (!\,\ell + 1); c_2.\mathtt{send}[0]())\}\,;\\
&\mathtt{let}\,\ell = \mathtt{ref}\,38\,\mathtt{in}\\
&c_0.\mathtt{send}[1](\ell); c_0.\mathtt{recv}[2];\\
&c_0.\mathtt{send}[1](\ell); c_0.\mathtt{recv}[2];\\
&\mathtt{let}\,x = !\,\ell\,\mathtt{in}\ \mathtt{assert}(x = 42)
\end{aligned}
$$

Roundtrip Reference Recursion Example

Roundtrip reference recursion program:

$$\begin{array}{l}
\mathbf{let}\,(c_0, c_1, c_2) = \mathbf{new_chan}\,3\,\mathbf{in} \\
\mathbf{fork}\ \{\mathtt{loop}(\mathbf{let}\,\ell = c_1.\mathbf{recv}[0]\,\mathbf{in}\,\ell \leftarrow (!\,\ell + 1); c_1.\mathbf{send}[2](\ell))\}\,; \\
\mathbf{fork}\ \{\mathtt{loop}(\mathbf{let}\,\ell = c_2.\mathbf{recv}[1]\,\mathbf{in}\,\ell \leftarrow (!\,\ell + 1); c_2.\mathbf{send}[0]())\}\,; \\
\mathbf{let}\,\ell = \mathbf{ref}\,38\,\mathbf{in} \\
c_0.\mathbf{send}[1](\ell); c_0.\mathbf{recv}[2]; \\
c_0.\mathbf{send}[1](\ell); c_0.\mathbf{recv}[2]; \\
\mathbf{let}\,x = !\,\ell\,\mathbf{in}\,\mathbf{assert}(x = 42)
\end{array}$$

Protocols:

$$\begin{array}{l}
c_0 \rightarrowtail \mu rec.\,\mathbf{!}\,[1]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{?}[2]\,\langle () \rangle \{\ell \mapsto (x+2)\}.\,rec \\
c_1 \rightarrowtail \mu rec.\,\mathbf{?}[0]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,[2]\,\langle \ell \rangle \{\ell \mapsto (x+1)\}.\,rec \\
c_2 \rightarrowtail \mu rec.\,\mathbf{?}[1]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,[0]\,\langle () \rangle \{\ell \mapsto (x+1)\}.\,rec
\end{array}$$

Roundtrip Reference Recursion and Routing Example

Roundtrip reference recursion and routing program:

$$
\begin{array}{l}
\mathbf{let}\,(c_0, c_1, c_2, c_3) = \mathbf{new\_chan}\,4\,\mathbf{in} \\
\mathbf{fork}\ \{\mathtt{loop}(\mathbf{let}\,v = c_1.\mathbf{recv}[0]\,\mathbf{in}\,\mathbf{let}\,b = c_1.\mathbf{recv}[0]\,\mathbf{in}\,c_1.\mathbf{send}[\mathbf{if}\,b\,\mathbf{then}\,2\,\mathbf{else}\,3](v))\}\,; \\
\mathbf{fork}\ \{\mathtt{loop}(\mathbf{let}\,\ell = c_2.\mathbf{recv}[1]\,\mathbf{in}\,\ell \leftarrow (!\,\ell + 2); c_2.\mathbf{send}[0](()))\}\,; \\
\mathbf{fork}\ \{\mathtt{loop}(\mathbf{let}\,\ell = c_3.\mathbf{recv}[1]\,\mathbf{in}\,\ell \leftarrow (!\,\ell + 2); c_3.\mathbf{send}[0](()))\}\,; \\
\mathbf{let}\,\ell = \mathbf{ref}\,38\,\mathbf{in} \\
c_0.\mathbf{send}[1](\ell); c_0.\mathbf{send}[1](\mathbf{true}); c_0.\mathbf{recv}[2]; \\
c_0.\mathbf{send}[1](\ell); c_0.\mathbf{send}[1](\mathbf{false}); c_0.\mathbf{recv}[3]; \\
\mathbf{let}\,x = !\,\ell\,\mathbf{in}\,\mathbf{assert}(x = 42)
\end{array}
$$

Roundtrip Reference Recursion and Routing Example

Roundtrip reference recursion and routing program:

$$
\begin{array}{l}
\mathbf{let}\,(c_0, c_1, c_2, c_3) = \mathbf{new\_chan}\,4\,\mathbf{in} \\
\mathbf{fork}\,\{\mathtt{loop}(\mathbf{let}\,v = c_1.\mathbf{recv}[0]\,\mathbf{in}\,\mathbf{let}\,b = c_1.\mathbf{recv}[0]\,\mathbf{in}\,c_1.\mathbf{send}[\mathbf{if}\,b\,\mathbf{then}\,2\,\mathbf{else}\,3](v))\}\,; \\
\mathbf{fork}\,\{\mathtt{loop}(\mathbf{let}\,\ell = c_2.\mathbf{recv}[1]\,\mathbf{in}\,\ell \leftarrow (!\,\ell + 2); c_2.\mathbf{send}[0](()))\}\,; \\
\mathbf{fork}\,\{\mathtt{loop}(\mathbf{let}\,\ell = c_3.\mathbf{recv}[1]\,\mathbf{in}\,\ell \leftarrow (!\,\ell + 2); c_3.\mathbf{send}[0](()))\}\,; \\
\mathbf{let}\,\ell = \mathbf{ref}\,38\,\mathbf{in} \\
c_0.\mathbf{send}[1](\ell); c_0.\mathbf{send}[1](\mathbf{true}); c_0.\mathbf{recv}[2]; \\
c_0.\mathbf{send}[1](\ell); c_0.\mathbf{send}[1](\mathbf{false}); c_0.\mathbf{recv}[3]; \\
\mathbf{let}\,x = !\,\ell\,\mathbf{in}\,\mathbf{assert}(x = 42)
\end{array}
$$

Protocols:

$$
\begin{array}{l}
c_0 \rightarrowtail \mu rec.\,\mathbf{!}\,[1]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,[1]\,(b : \mathbb{B})\,\langle b \rangle. \\
\qquad\qquad\qquad \mathbf{?}[\mathbf{if}\,b\,\mathbf{then}\,2\,\mathbf{else}\,3]\,\langle () \rangle \{\ell \mapsto (x + 2)\}.\,rec \\
c_1 \rightarrowtail \mu rec.\,\mathbf{?}[0]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{?}[0]\,(b : \mathbb{B})\,\langle b \rangle. \\
\qquad\qquad\qquad \mathbf{!}\,[\mathbf{if}\,b\,\mathbf{then}\,2\,\mathbf{else}\,3]\,\langle \ell \rangle \{\ell \mapsto x\}.\,rec \\
c_2 \rightarrowtail \mu rec.\,\mathbf{?}[2]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,[0]\,\langle () \rangle \{\ell \mapsto (x + 2)\}.\,rec \\
c_3 \rightarrowtail \mu rec.\,\mathbf{?}[2]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,[0]\,\langle () \rangle \{\ell \mapsto (x + 2)\}.\,rec
\end{array}
$$

Roundtrip Reference Recursion and Routing Example

Roundtrip reference recursion and routing program:

$$
\begin{array}{l}
\mathbf{let}\,(c_0, c_1, c_2, c_3) = \mathbf{new\_chan}\,4\,\mathbf{in} \\
\mathbf{fork}\,\{\mathtt{loop}(\mathbf{let}\,v = c_1.\mathbf{recv}[0]\,\mathbf{in}\,\mathbf{let}\,b = c_1.\mathbf{recv}[0]\,\mathbf{in}\,c_1.\mathbf{send}[\mathbf{if}\,b\,\mathbf{then}\,2\,\mathbf{else}\,3](v))\}\,; \\
\mathbf{fork}\,\{\mathtt{loop}(\mathbf{let}\,\ell = c_2.\mathbf{recv}[1]\,\mathbf{in}\,\ell \leftarrow (!\,\ell + 2); c_2.\mathbf{send}[0](()))\}\,; \\
\mathbf{fork}\,\{\mathtt{loop}(\mathbf{let}\,\ell = c_3.\mathbf{recv}[1]\,\mathbf{in}\,\ell \leftarrow (!\,\ell + 2); c_3.\mathbf{send}[0](()))\}\,; \\
\mathbf{let}\,\ell = \mathbf{ref}\,38\,\mathbf{in} \\
c_0.\mathbf{send}[1](\ell); c_0.\mathbf{send}[1](\mathbf{true}); c_0.\mathbf{recv}[2]; \\
c_0.\mathbf{send}[1](\ell); c_0.\mathbf{send}[1](\mathbf{false}); c_0.\mathbf{recv}[3]; \\
\mathbf{let}\,x = !\,\ell\,\mathbf{in}\,\mathbf{assert}(x = 42)
\end{array}
$$

Protocols:

$$
\begin{array}{ll}
c_0 \rightarrowtail \mu rec. & \mathbf{!}\,[1]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,[1]\,(b : \mathbb{B})\,\langle b \rangle. \\
& \mathbf{?}[\mathbf{if}\,b\,\mathbf{then}\,2\,\mathbf{else}\,3]\,\langle () \rangle \{\ell \mapsto (x+2)\}.\,rec \\
c_1 \rightarrowtail \mu rec. & \mathbf{?}[0]\,(v : \mathrm{Val})\,\langle v \rangle.\,\mathbf{?}[0]\,(b : \mathbb{B})\,\langle b \rangle. \\
& \mathbf{!}\,[\mathbf{if}\,b\,\mathbf{then}\,2\,\mathbf{else}\,3]\,\langle v \rangle.\,rec \\
c_2 \rightarrowtail \mu rec. & \mathbf{?}[2]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,[0]\,\langle () \rangle \{\ell \mapsto (x+2)\}.\,rec \\
c_3 \rightarrowtail \mu rec. & \mathbf{?}[2]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,\mathbf{!}\,[0]\,\langle () \rangle \{\ell \mapsto (x+2)\}.\,rec
\end{array}
$$

Case Study:
Chang and Roberts
Ring Leader Election

Chang and Roberts Ring Leader Election - Algorithm

Consider n actors, arranged in a ring

- Ex1: $0 \rightarrow 1, 1 \rightarrow 2, 2 \rightarrow 1$
- Ex2: $0 \rightarrow 2, 2 \rightarrow 1, 1 \rightarrow 0$

Chang and Roberts Ring Leader Election - Algorithm

Consider n actors, arranged in a ring

- Ex1: $0 \rightarrow 1$, $1 \rightarrow 2$, $2 \rightarrow 1$
- Ex2: $0 \rightarrow 2$, $2 \rightarrow 1$, $1 \rightarrow 0$

Actors are tagged as participating or not; everyone starts untagged

- Tag as participating whenever any message is sent

Chang and Roberts Ring Leader Election - Algorithm

Consider n actors, arranged in a ring

- Ex1: $0 \to 1, 1 \to 2, 2 \to 1$
- Ex2: $0 \to 2, 2 \to 1, 1 \to 0$

Actors are tagged as participating or not; everyone starts untagged

- Tag as participating whenever any message is sent

Messages types are election(i') **(1)** and elected(i') **(2)**

Chang and Roberts Ring Leader Election - Algorithm

Consider n actors, arranged in a ring

- Ex1: $0 \to 1, 1 \to 2, 2 \to 1$
- Ex2: $0 \to 2, 2 \to 1, 1 \to 0$

Actors are tagged as participating or not; everyone starts untagged

- Tag as participating whenever any message is sent

Messages types are election(i') **(1)** and elected(i') **(2)**
Received election(i') messages are compared to the receivers id i and

- If $i' > i$, send election(i') **(1.1)**
- If $i' = i$, we are elected, send elected(i) **(1.2)**
- If we are not participating, send election(i) **(1.3)**
- If we are already participating, do nothing **(1.4)**

Chang and Roberts Ring Leader Election - Algorithm

Consider n actors, arranged in a ring

- Ex1: $0 \rightarrow 1, 1 \rightarrow 2, 2 \rightarrow 1$
- Ex2: $0 \rightarrow 2, 2 \rightarrow 1, 1 \rightarrow 0$

Actors are tagged as participating or not; everyone starts untagged

- Tag as participating whenever any message is sent

Messages types are election(i') **(1)** and elected(i') **(2)**

Received election(i') messages are compared to the receivers id i and

- If $i' > i$, send election(i') **(1.1)**
- If $i' = i$, we are elected, send elected(i) **(1.2)**
- If we are not participating, send election(i) **(1.3)**
- If we are already participating, do nothing **(1.4)**

Received elected(i') messages are compared to the participants id i and

- If $i' = i$, terminate by returning i' **(2.1)**
- If $i' \neq i$, send elected(i'), and terminate by returning i' **(2.2)**

Chang and Roberts Ring Leader Election - Implementation

We encode election(i) as $\mathtt{inl}\ i$ and elected(i) as $\mathtt{inr}\ i$.
The leader election process can then be implemented as follows:

$$
\begin{array}{ll}
\text{process } c\ il\ i\ ir \triangleq \mathtt{rec}\ rec\ isp = & \\
\quad \mathtt{match}\ c.\mathtt{recv}[ir]\ \mathtt{with} & \\
\quad \mid \mathtt{inl}\ i' \Rightarrow \mathtt{if}\ i < i'\ \mathtt{then}\ c.\mathtt{send}[il](\mathtt{inl}\ i');\ rec\ \mathtt{true} & (\mathbf{1.1}) \\
\qquad\qquad\quad \mathtt{else}\ \mathtt{if}\ i = i'\ \mathtt{then}\ c.\mathtt{send}[il](\mathtt{inr}\ i);\ rec\ \mathtt{false} & (\mathbf{1.2}) \\
\qquad\qquad\quad \mathtt{else}\ \mathtt{if}\ isp\ \mathtt{then}\ rec\ \mathtt{true} & (\mathbf{1.3}) \\
\qquad\qquad\quad \mathtt{else}\ c.\mathtt{send}[il](\mathtt{inl}\ i);\ rec\ \mathtt{true} & (\mathbf{1.4}) \\
\quad \mid \mathtt{inr}\ i' \Rightarrow \mathtt{if}\ i = i'\ \mathtt{then}\ i' & (\mathbf{2.1}) \\
\qquad\qquad\quad \mathtt{else}\ c.\mathtt{send}[il](\mathtt{inr}\ i');\ i' & (\mathbf{2.2}) \\
\quad \mathtt{end} &
\end{array}
$$

Chang and Roberts Ring Leader Election - Protocol

We denote branching protocols as (implemented in terms of receive protocols):

$$\&[i] \begin{Bmatrix} \mathtt{inl}(\vec{x_1} : \vec{\tau_1})\langle v_1 \rangle\{P_1\} \Rightarrow p_1 \\ \mathtt{inr}(\vec{x_2} : \vec{\tau_2})\langle v_2 \rangle\{P_2\} \Rightarrow p_2 \end{Bmatrix}$$

We can then define the ring leader election protocol as:

$$\begin{aligned}
&\mathsf{rle_prot}(il\ i\ ir : \mathbb{N})(p : \mathbb{N} \to \mathsf{iProto}) : \mathsf{iProto} \triangleq \mu rec.\ \lambda isp. \\
&\&[ir] \left\{ \begin{array}{llr}
\mathtt{inl}(i' : \mathbb{N})\langle i' \rangle \Rightarrow & \mathtt{if}\ i < i'\ \mathtt{then}\ !\,[il]\ \langle \mathtt{inl}\ i' \rangle .\ rec\ \mathtt{true} & \mathbf{(1.1)} \\
& \mathtt{else\ if}\ i = i'\ \mathtt{then}\ !\,[il]\ \langle \mathtt{inr}\ i \rangle .\ rec\ \mathtt{false} & \mathbf{(1.2)} \\
& \mathtt{else\ if}\ isp\ \mathtt{then}\ rec\ \mathtt{true} & \mathbf{(1.3)} \\
& \mathtt{else}\ !\,[il]\ \langle \mathtt{inl}\ i \rangle .\ rec\ \mathtt{true} & \mathbf{(1.4)} \\
\mathtt{inr}(i' : \mathbb{N})\langle i' \rangle \Rightarrow & \mathtt{if}\ i = i'\ \mathtt{then}\ p\ i & \mathbf{(2.1)} \\
& \mathtt{else}\ !\,[il]\ \langle \mathtt{inr}\ i' \rangle .\ p\ i' & \mathbf{(2.2)}
\end{array} \right\}
\end{aligned}$$

This lets us verify the ring leader process:

$$\{c \rightarrowtail (\mathsf{rle\_prot}\ il\ i\ ir\ p\ isp)\}\ \mathsf{process}\ c\ il\ i\ ir\ isp\ \{v.\ \exists (i' : \mathbb{N}).\ v = i' * c \rightarrowtail (p\ i')\}$$

Chang and Roberts Ring Leader Election - Verification

We test leader agreement with a central coordinator as follows:

$$\begin{array}{l}
\mathtt{let}\,(c_0, c_1, c_2, c_3) = \mathtt{new_chan}\; 4\; \mathtt{in} \\
\mathtt{fork}\;\{c_1.\mathtt{send}[2](\mathtt{inl}\; 1); \mathtt{let}\; i = \text{process}\; c_1\; 2\; 1\; 3\; \mathtt{true}\; \mathtt{in}\; c_1.\mathtt{send}[0](i)\}\; ; \\
\mathtt{fork}\;\{c_2.\mathtt{send}[3](\mathtt{inl}\; 2); \mathtt{let}\; i = \text{process}\; c_2\; 3\; 2\; 1\; \mathtt{true}\; \mathtt{in}\; c_2.\mathtt{send}[0](i)\}\; ; \\
\mathtt{fork}\;\{\mathtt{let}\; i = \text{process}\; c_3\; 1\; 3\; 2\; \mathtt{false}\; \mathtt{in}\; c_3.\mathtt{send}[0](i)\}\; ; \\
\mathtt{let}\; res_1 = c_0.\mathtt{recv}[1]\; \mathtt{in} \\
\mathtt{let}\; res_2 = c_0.\mathtt{recv}[2]\; \mathtt{in} \\
\mathtt{let}\; res_3 = c_0.\mathtt{recv}[3]\; \mathtt{in} \\
\mathtt{assert}(res_1 = res_2); \mathtt{assert}(res_2 = res_3).
\end{array}$$

Chang and Roberts Ring Leader Election - Verification

We test leader agreement with a central coordinator as follows:

$$
\begin{array}{l}
\mathbf{let}\,(c_0, c_1, c_2, c_3) = \mathbf{new\_chan}\; 4\; \mathbf{in} \\
\mathbf{fork}\ \{c_1.\mathbf{send}[2](\mathbf{inl}\; 1); \mathbf{let}\; i = \text{process}\; c_1\; 2\; 1\; 3\; \mathbf{true}\; \mathbf{in}\; c_1.\mathbf{send}[0](i)\}\;; \\
\mathbf{fork}\ \{c_2.\mathbf{send}[3](\mathbf{inl}\; 2); \mathbf{let}\; i = \text{process}\; c_2\; 3\; 2\; 1\; \mathbf{true}\; \mathbf{in}\; c_2.\mathbf{send}[0](i)\}\;; \\
\mathbf{fork}\ \{\mathbf{let}\; i = \text{process}\; c_3\; 1\; 3\; 2\; \mathbf{false}\; \mathbf{in}\; c_3.\mathbf{send}[0](i)\}\;; \\
\mathbf{let}\; res_1 = c_0.\mathbf{recv}[1]\; \mathbf{in} \\
\mathbf{let}\; res_2 = c_0.\mathbf{recv}[2]\; \mathbf{in} \\
\mathbf{let}\; res_3 = c_0.\mathbf{recv}[3]\; \mathbf{in} \\
\mathbf{assert}(res_1 = res_2); \mathbf{assert}(res_2 = res_3).
\end{array}
$$

Protocols:

$$
\begin{array}{l}
c_0 \rightarrowtail \mathbf{?}[1]\,(i : \mathbb{N})\,\langle i \rangle.\, \mathbf{?}[2]\,\langle i \rangle.\, \mathbf{?}[3]\,\langle i \rangle.\, \mathbf{end} \\
c_1 \rightarrowtail \mathbf{!}\,[2]\,\langle \mathbf{inl}\; 1 \rangle.\, \text{rle\_prot}\; 2\; 1\; 3\; (\lambda i.\, \mathbf{!}\,[0]\,\langle i \rangle.\, \mathbf{end})\; \mathbf{true} \\
c_2 \rightarrowtail \mathbf{!}\,[3]\,\langle \mathbf{inl}\; 2 \rangle.\, \text{rle\_prot}\; 3\; 2\; 1\; (\lambda i.\, \mathbf{!}\,[0]\,\langle i \rangle.\, \mathbf{end})\; \mathbf{true} \\
c_3 \rightarrowtail \text{rle\_prot}\; 1\; 3\; 2\; (\lambda i.\, \mathbf{!}\,[0]\,\langle i \rangle.\, \mathbf{end})\; \mathbf{false}
\end{array}
$$

Chang and Roberts Ring Leader Election - Verification

We test leader agreement with a central coordinator as follows:

$$
\begin{array}{l}
\mathtt{let}\,(c_0, c_1, c_2, c_3) = \mathtt{new\_chan}\ 4\ \mathtt{in} \\
\mathtt{fork}\ \{c_1.\mathtt{send}[2](\mathtt{inl}\ 1);\ \mathtt{let}\ i = \mathsf{process}\ c_1\ 2\ 1\ 3\ \mathtt{true}\ \mathtt{in}\ c_1.\mathtt{send}[0](i)\}\ ; \\
\mathtt{fork}\ \{c_2.\mathtt{send}[3](\mathtt{inl}\ 2);\ \mathtt{let}\ i = \mathsf{process}\ c_2\ 3\ 2\ 1\ \mathtt{true}\ \mathtt{in}\ c_2.\mathtt{send}[0](i)\}\ ; \\
\mathtt{fork}\ \{\mathtt{let}\ i = \mathsf{process}\ c_3\ 1\ 3\ 2\ \mathtt{false}\ \mathtt{in}\ c_3.\mathtt{send}[0](i)\}\ ; \\
\mathtt{let}\ res_1 = c_0.\mathtt{recv}[1]\ \mathtt{in} \\
\mathtt{let}\ res_2 = c_0.\mathtt{recv}[2]\ \mathtt{in} \\
\mathtt{let}\ res_3 = c_0.\mathtt{recv}[3]\ \mathtt{in} \\
\mathtt{assert}(res_1 = res_2);\ \mathtt{assert}(res_2 = res_3).
\end{array}
$$

Protocols:

$$
\begin{array}{l}
c_0 \rightarrowtail\ ?[1]\,(i : \mathbb{N})\,\langle i \rangle.\ ?[2]\,\langle i \rangle.\ ?[3]\,\langle i \rangle.\ \mathbf{end} \\
c_1 \rightarrowtail\ ![0]\,\langle i \rangle.\ \mathbf{end} \\
c_2 \rightarrowtail\ ![0]\,\langle i \rangle.\ \mathbf{end} \\
c_3 \rightarrowtail\ ![0]\,\langle i \rangle.\ \mathbf{end}
\end{array}
$$

Language Parametricity of Multiparty Actris

Protocol consistency

We have generically defined protocol consistency in separation logic as follows:

$$\frac{(\forall i, j.\ \texttt{semantic\_dual}\ ps\ i\ j)}{\texttt{prot\_consistent}\ ps}$$

$$\frac{\begin{array}{c} ps[i] = \mathbf{!}\,[j]\,\vec{x}:\vec{\tau}\,\langle v_1\rangle\{P_1\}.\,p_1 \mathrel{-\!\!*} ps[j] = \mathbf{?}[i]\,\vec{y}:\vec{\sigma}\,\langle v_2\rangle\{P_2\}.\,p_2 \mathrel{-\!\!*} \\ \forall\vec{x}:\vec{\tau}.\,P_1 \mathrel{-\!\!*} \exists\vec{y}:\vec{\sigma}.\,v_1 = v_2 * P_2 * \triangleright(\texttt{prot\_consistent}\ (ps[i := p_1][j := p_2])) \end{array}}{\texttt{semantic\_dual}\ ps\ i\ j}$$

Multiparty Actris Ghost Theory

We prove language-generic ghost theory rules:

PROTO-ALLOC

$$\frac{\texttt{prot\_consistent}\ ps}{\Rrightarrow \exists \chi.\ \mathsf{prot\_ctx}\ \chi * \mathop{\Huge\ast}_{i \mapsto p \in ps} \mathsf{prot\_own}\ \chi\ i\ p}$$

PROTO-STEP

$$\frac{\mathsf{prot\_ctx}\ \chi \qquad P_1[\vec{t}/\vec{x}] \qquad \mathsf{prot\_own}\ \chi\ i\ (\mathbf{!}\,[j]\,\vec{x_1} : \vec{\tau_1}\ \langle v_1 \rangle \{P_1\}.\, p_1) \qquad \mathsf{prot\_own}\ \chi\ j\ (\mathbf{?}[i]\,\vec{x_2} : \vec{\tau_2}\ \langle v_2 \rangle \{P_2\}.\, p_2)}{\Rrightarrow \triangleright \exists \vec{y}.\ \mathsf{prot\_ctx}\ \chi * \mathsf{prot\_own}\ \chi\ i\ (p_1[\vec{t}/\vec{x_1}]) * \mathsf{prot\_own}\ \chi\ j\ (p_2[\vec{y}/\vec{x_2}]) * P_2[\vec{y}/\vec{x_2}]}$$

One can then prove the Hoare triples of a language (such as Ht-send, Ht-recv, and Ht-new), using the ghost theory.

Conclusion and Future Work

Conclusion and Future Work

Recover (Binary) Actris features

- Asynchronous communication
- Subprotocols

Better methodology for proving protocol consistency

- Abstraction and Modularity
- Automation via Model Checking?

Guarantee deadlock freedom

- Leverage connectivity graphs

Multiparty Actris for distributed systems

- Leverage Aneris

$$
\begin{aligned}
&!\,[1]\,\langle\text{“Thank you”}\rangle\{\texttt{ActrisKnowledge}\}.\\
&\mu \mathit{rec}.\ ?[1]\,(q : \texttt{Question i})\,\langle q\rangle\{\texttt{AboutMDSP}\ q\}.\\
&\qquad !\,[i]\,(a : \texttt{Answer})\,\langle a\rangle\{\texttt{Insightful}\ q\ a\}.\,\mathit{rec}
\end{aligned}
$$