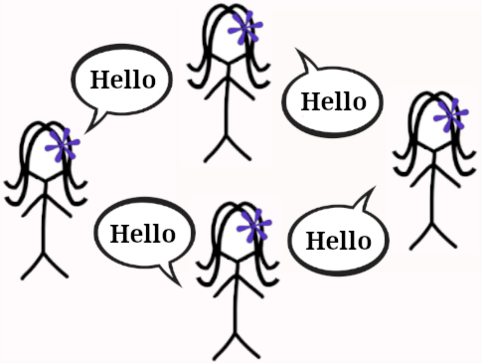
Multris:
Functional Verification of Multiparty Message Passing in Separation Logic

Jonas Kastberg Hinrichsen
IT University of Copenhagen

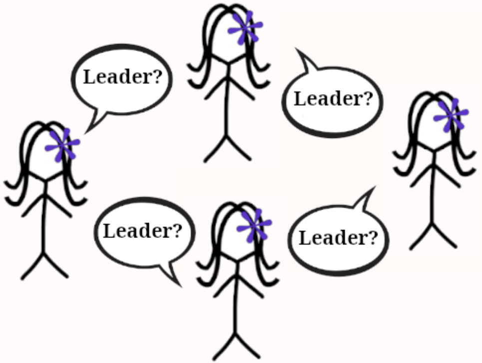
Jules Jacobs
Cornell University

Robbert Krebbers
Radboud University Nijmegen

Multiparty Message Passing



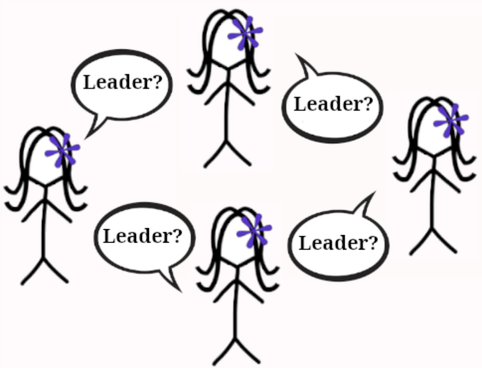
Multiparty Message Passing



Multiparty Message Passing

Multiparty message passing

- Message passing with dependent interactions between multiple parties



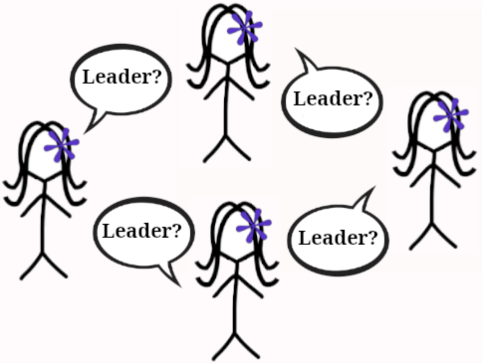
Multiparty Message Passing

Multiparty message passing

- Message passing with dependent interactions between multiple parties

Hard to get right

- Concurrency is hard



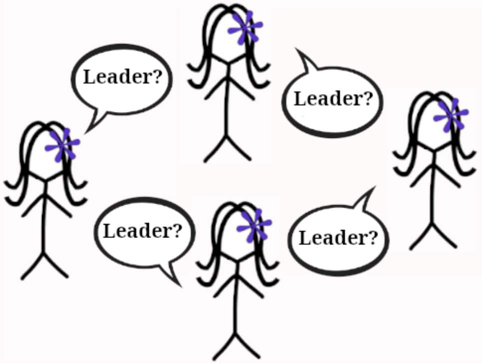
Multiparty Message Passing

Multiparty message passing

- Message passing with dependent interactions between multiple parties

Hard to get right

- Concurrency is hard
- Especially in conjunction with other implementation details
  - e.g. shared memory, higher-order functions, recursion



Multiparty Message Passing

Multiparty message passing

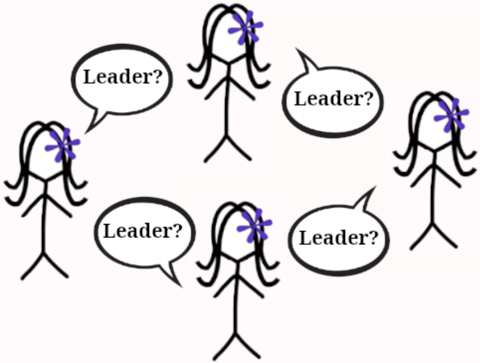
- Message passing with dependent interactions between multiple parties

Hard to get right

- Concurrency is hard
- Especially in conjunction with other implementation details
  - e.g. shared memory, higher-order functions, recursion

Warrants functional verification

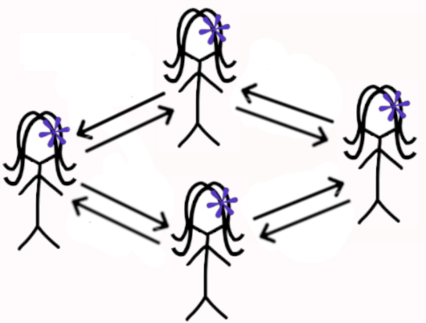
- No results that supports all the above
- We want validation in a mechanised theorem prover



Our Setting of Multiparty Message Passing

Message passing over bi-directional channels with distinct channel endpoints

- Each endpoint correspond to one party



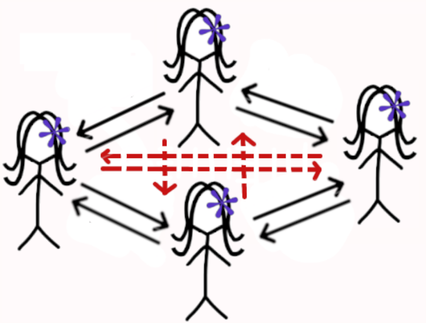
Our Setting of Multiparty Message Passing

Message passing over bi-directional channels with distinct channel endpoints

- Each endpoint correspond to one party

Channel endpoints are fully connected, indexed by communicating parties

- For simplicity sake



Our Setting of Multiparty Message Passing

Message passing over bi-directional channels with distinct channel endpoints

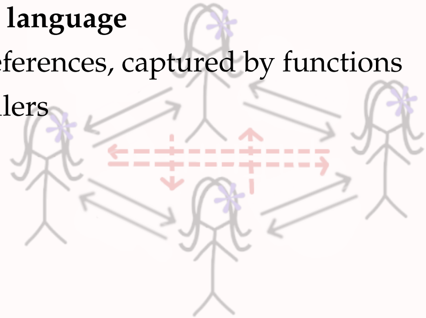
- Each endpoint correspond to one party

Channel endpoints are fully connected, indexed by communicating parties

- For simplicity sake

Channel endpoints are first class citizens of the language

- Can be passed around as values, stored in references, captured by functions
- Similar to Go channels and BSD socket handlers



Our Setting of Multiparty Message Passing

Message passing over bi-directional channels with distinct channel endpoints

- Each endpoint correspond to one party

Channel endpoints are fully connected, indexed by communicating parties

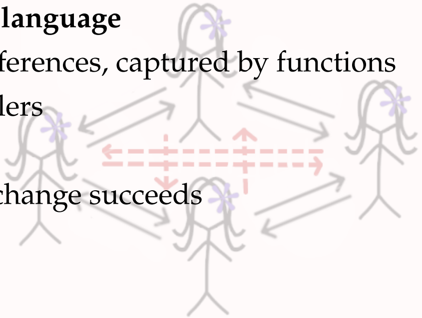
- For simplicity sake

Channel endpoints are first class citizens of the language

- Can be passed around as values, stored in references, captured by functions
- Similar to Go channels and BSD socket handlers

Synchronous exchanges

- Attempted sends and receives block until exchange succeeds



Our Setting of Multiparty Message Passing

Message passing over bi-directional channels with distinct channel endpoints

- Each endpoint correspond to one party

Channel endpoints are fully connected, indexed by communicating parties

- For simplicity sake

Channel endpoints are first class citizens of the language

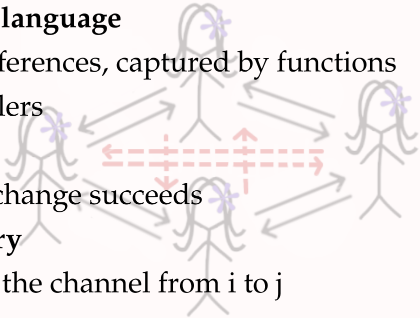
- Can be passed around as values, stored in references, captured by functions
- Similar to Go channels and BSD socket handlers

Synchronous exchanges

- Attempted sends and receives block until exchange succeeds

Implementation via references in shared memory

- Implemented as an N x N matrix where i,j is the channel from i to j



Multiparty Message Passing in Shared Memory

Multiparty channels API:

new_chan(n)	Creates a multiparty channel with n parties, returning a tuple $(c_0, \ldots, c_{(n-1)})$ of endpoints
$c_i[j]$.**send**(v)	Sends a value v via endpoint c_i to party j (synchronously)
$c_i[j]$.**recv**()	Receives a value via endpoint c_i from party j

Multiparty Message Passing in Shared Memory

Multiparty channels API:

$\mathbf{new\_chan}(n)$	Creates a multiparty channel with n parties, returning a tuple $(c_0, \ldots, c_{(n-1)})$ of endpoints
$c_i[j].\mathbf{send}(v)$	Sends a value v via endpoint c_i to party j (synchronously)
$c_i[j].\mathbf{recv}()$	Receives a value via endpoint c_i from party j

Example program: Roundtrip

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

Safety and Functional Correctness

Example program: Roundtrip

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

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

Safety and Functional Correctness

Example program: Roundtrip

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

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

	Safety via type systems	**Functional Correctness** via separation logic

Safety and Functional Correctness

Example program: Roundtrip

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

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

	Safety via type systems	**Functional Correctness** via separation logic
Multiparty	Multiparty Session Types	???

Safety and Functional Correctness

Example program: Roundtrip

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

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

$$c_0 : {!}[1]\mathbb{Z}.\, {?}[2]\mathbb{Z}.\, \textbf{end} \qquad c_1 : {?}[0]\mathbb{Z}.\, {!}[2]\mathbb{Z}.\, \textbf{end} \qquad c_2 : {?}[1]\mathbb{Z}.\, {!}[0]\mathbb{Z}.\, \textbf{end}$$

	Safety via type systems	**Functional Correctness** via separation logic
Multiparty	Multiparty Session Types	???

! is send, **?** is receive

Safety and Functional Correctness

Example program: Roundtrip

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

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

$$c_0 : \mathbf{!}[1]\mathbb{Z}.\ \mathbf{?}[2]\mathbb{Z}.\ \mathbf{end} \qquad c_1 : \mathbf{?}[0]\mathbb{Z}.\ \mathbf{!}[2]\mathbb{Z}.\ \mathbf{end} \qquad c_2 : \mathbf{?}[1]\mathbb{Z}.\ \mathbf{!}[0]\mathbb{Z}.\ \mathbf{end}$$

	Safety via type systems	**Functional Correctness** via separation logic
Multiparty	Multiparty Session Types	???
Binary	Session Types	Dependent separation protocols

! is send, **?** is receive

Key Idea

Prior work:

- **Safety (Multiparty):** $\mathbf{!}[1]\mathbb{Z}.\,\mathbf{?}[2]\mathbb{Z}.\,\textbf{end}$

Key Idea

Prior work:

- **Safety (Multiparty):** $![1]\mathbb{Z}.\ ?[2]\mathbb{Z}.\ \textbf{end}$
- **Safety (Binary):** $!\mathbb{Z}.\ ?\mathbb{Z}.\ \textbf{end}$

Key Idea

Prior work: Dependent separation protocols (DSPs)

- **Safety (Multiparty):** $![1]\mathbb{Z}.\ ?[2]\mathbb{Z}.\ \textbf{end}$
- **Safety (Binary):** $!\mathbb{Z}.\ ?\mathbb{Z}.\ \textbf{end}$
- **Functional Correctness (Binary):** $!\langle 40 \rangle.\ ?\langle 42 \rangle.\ \textbf{end}$

Key Idea

Prior work: Dependent separation protocols (DSPs)

- **Safety (Multiparty):** $![1]\mathbb{Z}.\ ?[2]\mathbb{Z}.\ \textsf{end}$
- **Safety (Binary):** $!\mathbb{Z}.\ ?\mathbb{Z}.\ \textsf{end}$
- **Functional Correctness (Binary):** $!\,(x : \mathbb{Z})\ \langle x \rangle.\ ?\langle x + 2 \rangle.\ \textsf{end}$

Key Idea

Prior work: Dependent separation protocols (DSPs)

- **Safety (Multiparty):** $!\,[1]\mathbb{Z}.\ ?[2]\mathbb{Z}.\ \textbf{end}$
- **Safety (Binary):** $!\mathbb{Z}.\ ?\mathbb{Z}.\ \textbf{end}$
- **Functional Correctness (Binary):** $!\,(x : \mathbb{Z})\ \langle x \rangle.\ ?\langle x + 2 \rangle.\ \textbf{end}$
  - **Actris:** Rich separation logic based on DSPs in Rocq (via Iris)

Key Idea

Prior work: Dependent separation protocols (DSPs)

- **Safety (Multiparty):** $![1]\mathbb{Z}.\,?[2]\mathbb{Z}.\,\textbf{end}$
- **Safety (Binary):** $!\mathbb{Z}.\,?\mathbb{Z}.\,\textbf{end}$
- **Functional Correctness (Binary):** $!\,(x:\mathbb{Z})\,\langle x\rangle.\,?\langle x+2\rangle.\,\textbf{end}$
  - **Actris:** Rich separation logic based on DSPs in Rocq (via Iris)

Key Idea: Multiparty dependent separation protocols! (MDSPs)

- **Functional Correctness (Multiparty):** $!\,[1]\,(x:\mathbb{Z})\,\langle x\rangle.\,?[2]\,\langle x+2\rangle.\,\textbf{end}$

Key Idea

Prior work: Dependent separation protocols (DSPs)

- **Safety (Multiparty):** $!{[1]}\mathbb{Z}.\,?{[2]}\mathbb{Z}.\,\textbf{end}$
- **Safety (Binary):** $!\mathbb{Z}.\,?\mathbb{Z}.\,\textbf{end}$
- **Functional Correctness (Binary):** $!\,(x : \mathbb{Z})\,\langle x\rangle.\,?\langle x + 2\rangle.\,\textbf{end}$
  - **Actris:** Rich separation logic based on DSPs in Rocq (via Iris)

Key Idea: Multiparty dependent separation protocols! (MDSPs)

- **Functional Correctness (Multiparty):** $!\,[1]\,(x : \mathbb{Z})\,\langle x\rangle.\,?[2]\,\langle x + 2\rangle.\,\textbf{end}$
  - **Multris:** Rich separation logic based on MDSPs in Rocq (via Iris)

Key Idea

Prior work: Dependent separation protocols (DSPs)

- **Safety (Multiparty):** $![1]\mathbb{Z}.\,?[2]\mathbb{Z}.\,\mathbf{end}$
- **Safety (Binary):** $!\mathbb{Z}.\,?\mathbb{Z}.\,\mathbf{end}$
- **Functional Correctness (Binary):** $!\,(x : \mathbb{Z})\,\langle x\rangle.\,?\langle x+2\rangle.\,\mathbf{end}$
  - **Actris:** Rich separation logic based on DSPs in Rocq (via Iris)

Key Idea: Multiparty dependent separation protocols! (MDSPs)

- **Functional Correctness (Multiparty):** $!\,[1]\,(x : \mathbb{Z})\,\langle x\rangle.\,?[2]\,\langle x+2\rangle.\,\mathbf{end}$
  - **Multris:** Rich separation logic based on MDSPs in Rocq (via Iris)

Example Program: Roundtrip

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

Key Idea

Prior work: Dependent separation protocols (DSPs)

- **Safety (Multiparty):** $![1]\mathbb{Z}.\ ?[2]\mathbb{Z}.\ \textbf{end}$
- **Safety (Binary):** $!\mathbb{Z}.\ ?\mathbb{Z}.\ \textbf{end}$
- **Functional Correctness (Binary):** $!\,(x : \mathbb{Z})\ \langle x \rangle.\ ?\langle x + 2 \rangle.\ \textbf{end}$
  - **Actris:** Rich separation logic based on DSPs in Rocq (via Iris)

Key Idea: Multiparty dependent separation protocols! (MDSPs)

- **Functional Correctness (Multiparty):** $!\,[1]\,(x : \mathbb{Z})\ \langle x \rangle.\ ?[2]\ \langle x + 2 \rangle.\ \textbf{end}$
  - **Multris:** Rich separation logic based on MDSPs in Rocq (via Iris)

Example Program: Roundtrip

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

Remaining challenge: How to guarantee consistent global communication?

Global Protocol Consistency

Remaining challenge: How to guarantee consistent global communication?

$$\texttt{let}\,(c_0, c_1, c_2) = \texttt{new\_chan}(3)\,\texttt{in}$$

$$\left(\begin{array}{l} \texttt{let}\,x = 40\,\texttt{in}\,c_0[1].\texttt{send}(x); \\ \texttt{assert}(c_0[2].\texttt{recv}() = x + 2) \end{array} \middle\| \begin{array}{l} \texttt{let}\,y = c_1[0].\texttt{recv}()\,\texttt{in} \\ c_1[2].\texttt{send}(y + 1) \end{array} \middle\| \begin{array}{l} \texttt{let}\,z = c_2[1].\texttt{recv}()\,\texttt{in} \\ c_2[0].\texttt{send}(z + 1) \end{array}\right)$$

Global Protocol Consistency

Remaining challenge: How to guarantee consistent global communication?

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

Prior work: Syntactic duality

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

Global Protocol Consistency

Remaining challenge: How to guarantee consistent global communication?

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

Prior work: Syntactic duality

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

This work:

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

Global Protocol Consistency

Remaining challenge: How to guarantee consistent global communication?

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

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:

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

Global Protocol Consistency

Remaining challenge: How to guarantee consistent global communication?

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

Prior work: Syntactic duality

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

This work:

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

Global Protocol Consistency

Remaining challenge: How to guarantee consistent global communication?

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

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

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

Global Protocol Consistency

Remaining challenge: How to guarantee consistent global communication?

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

Prior work: Syntactic duality

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

This work: Semantic duality

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

Key Idea: Define and prove consistency via separation logic!

Contributions

Multiparty dependent separation protocols (MDSPs)

- Rich specification language for describing multiparty message passing
- Protocol consistency defined in terms of semantic duality, proven in separation logic

Multris separation logic

- Separation logic for verifying multiparty communication via MDSPs
- Support for language-parametric instantiation of Multris

Verification of suite of multiparty programs

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

Full mechanisation in Rocq

- With tactic support for protocol consistency and channel primitives

Roadmap of this talk

Separation Logic Primer

- Operational semantics
- Hoare triples
- Separation logic

Tour of the Multris separation logic

- Multiparty dependent separation protocols and protocol consistency
- Verification rules for multiparty channels
- Verification of suite of roundtrip variations

Conclusion and Future Work

Separation Logic Primer

Operational Semantics

HeapLang: Untyped OCaml-like language

$$
\begin{aligned}
v, w \in \mathsf{Val} &::= z \mid \mathbf{true} \mid \mathbf{false} \mid () \mid \ell \mid \lambda x.\, e \\
e \in \mathsf{Expr} &::= v \mid x \mid e_1\ e_2 \mid \mathbf{let}\, x = e_1\ \mathbf{in}\, e_2 \mid e_1; e_2 \mid \\
&\quad \mathbf{ref}\, e \mid !\, e \mid e_1 \leftarrow e_2 \mid \\
&\quad (e_1 \parallel e_2) \mid \mathbf{assert}(e) \mid \ldots
\end{aligned}
$$

Operational Semantics

HeapLang: Untyped OCaml-like language

$$
\begin{aligned}
v, w \in \mathsf{Val} &::= z \mid \mathbf{true} \mid \mathbf{false} \mid () \mid \ell \mid \lambda x.\, e \\
e \in \mathsf{Expr} &::= v \mid x \mid e_1\ e_2 \mid \mathbf{let}\ x = e_1\ \mathbf{in}\ e_2 \mid e_1 ; e_2 \mid \\
&\quad \mathbf{ref}\ e \mid !\, e \mid e_1 \leftarrow e_2 \mid \\
&\quad (e_1 \parallel e_2) \mid \mathbf{assert}(e) \mid \ldots
\end{aligned}
$$

Example program:

$$
\begin{array}{l}
\mathbf{let}\ \ell_1 = \mathbf{ref}\ 0\ \mathbf{in} \\
\mathbf{let}\ \ell_2 = \mathbf{ref}\ 0\ \mathbf{in} \\
\big(\ell_1 \leftarrow !\,\ell_1 + 2 \parallel \ell_2 \leftarrow !\,\ell_2 + 2\big)\,; \\
\mathbf{assert}(!\,\ell_1 + !\,\ell_2 = 4)
\end{array}
$$

Operational Semantics

HeapLang: Untyped OCaml-like language

$$\begin{aligned} v, w \in \mathsf{Val} &::= z \mid \mathbf{true} \mid \mathbf{false} \mid () \mid \ell \mid \lambda x.\, e \\ e \in \mathsf{Expr} &::= v \mid x \mid e_1\, e_2 \mid \mathbf{let}\, x = e_1\, \mathbf{in}\, e_2 \mid e_1 ; e_2 \mid \\ &\quad\ \mathbf{ref}\, e \mid !\, e \mid e_1 \leftarrow e_2 \mid \\ &\quad\ (e_1 \parallel e_2) \mid \mathbf{assert}(e) \mid \ldots \end{aligned}$$

Example program:

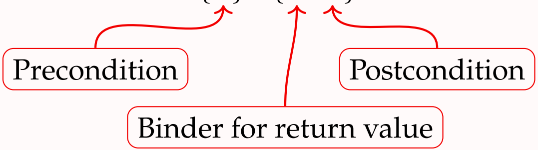
$$\begin{array}{l} \mathbf{let}\, \ell_1 = \mathbf{ref}\, 0\, \mathbf{in} \\ \mathbf{let}\, \ell_2 = \mathbf{ref}\, 0\, \mathbf{in} \\ \big(\ell_1 \leftarrow !\, \ell_1 + 2 \parallel \ell_2 \leftarrow !\, \ell_2 + 2\big)\, ; \\ \mathbf{assert}(!\, \ell_1 + !\, \ell_2 = 4) \end{array}$$

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

Hoare Triples

Hoare triples for partial functional correctness:

$$\{P\}\ e\ \{w.\ Q\}$$



If the initial state satisfies P, then:

- **Safety:** e does not crash
- **Postcondition validity:** if e terminates with value v, then the final state satisfies $Q[v/w]$

Separation Logic

Separation logic: propositions assert ownership and knowledge about the state

The points-to connective: $\ell \mapsto v$

- Provides the knowledge that location ℓ has value v, and
- Provides exclusive ownership of ℓ

Separating conjunction: $P * Q$ captures that the state consists of disjoint parts satisfying P and Q.

Separation Logic

Separation logic: propositions assert ownership and knowledge about the state

The points-to connective: $\ell \mapsto v$

- Provides the knowledge that location ℓ has value v, and
- Provides exclusive ownership of ℓ

Separating conjunction: $P * Q$ captures that the state consists of disjoint parts satisfying P and Q.

Enables modular reasoning, through disjointness:

Ht-frame

$$\frac{\{P\}\, e\, \{w.\, Q\}}{\{P * R\}\, e\, \{w.\, Q * R\}}$$

Hoare Triples for Seperation Logic

Hoare triples for references:

Ht-alloc
$\{\mathsf{True}\}\ \mathtt{ref}\, v\ \{\ell.\ \ell \mapsto v\}$

Ht-load
$\{\ell \mapsto v\}\ !\,\ell\ \{w.\ w = v * \ell \mapsto v\}$

Ht-store
$\{\ell \mapsto v\}\ \ell \leftarrow w\ \{\ell \mapsto w\}$

Hoare Triples for Seperation Logic

Hoare triples for references:

HT-ALLOC
$\{\mathsf{True}\}\ \mathtt{ref}\, v\ \{\ell.\ \ell \mapsto v\}$

HT-LOAD
$\{\ell \mapsto v\}\ !\,\ell\ \{w.\ w = v * \ell \mapsto v\}$

HT-STORE
$\{\ell \mapsto v\}\ \ell \leftarrow w\ \{\ell \mapsto w\}$

Hoare triples for structural expressions:

HT-LET
$$\frac{\{P\}\ e_1\ \{w_1.\ Q\} \qquad \forall w_1.\ \{Q\}\ e_2[w_1/x]\ \{w_2.\ R\}}{\{P\}\ \mathtt{let}\, x = e_1\ \mathtt{in}\, e_2\ \{w_2.\ R\}}$$

HT-ASSERT
$$\frac{\{P\}\ e\ \{w.\ w = \mathtt{true} * Q\}}{\{P\}\ \mathtt{assert}(e)\ \{Q\}}$$

HT-SEQ
$$\frac{\{P\}\ e_1\ \{w_1.\ Q\} \qquad \forall w_1.\ \{Q\}\ e_2\ \{w_2. R\}}{\{P\}\ e_1; e_2\ \{w_2.\ R\}}$$

HT-PAR
$$\frac{\{P_1\}\ e_1\ \{Q_1\} \qquad \{P_2\}\ e_2\ \{Q_2\}}{\{P_1 * P_2\}\ (e_1 \parallel e_2)\ \{Q_1 * Q_2\}}$$

Example Program - Verified

$$\mathbf{let}\ \ell_1 = \mathbf{ref}\ 0\ \mathbf{in}$$
$$\mathbf{let}\ \ell_2 = \mathbf{ref}\ 0\ \mathbf{in}$$
$$\big(\ell_1 \leftarrow !\,\ell_1 + 2 \parallel \ell_2 \leftarrow !\,\ell_2 + 2\big)\,;$$
$$\mathbf{assert}(!\,\ell_1 + !\,\ell_2 = 4)$$

Example Program - Verified

$$\{\mathsf{True}\}$$
$$\mathbf{let}\ \ell_1 = \mathbf{ref}\ 0\ \mathbf{in}$$
$$\mathbf{let}\ \ell_2 = \mathbf{ref}\ 0\ \mathbf{in}$$
$$(\ell_1 \leftarrow !\,\ell_1 + 2 \parallel \ell_2 \leftarrow !\,\ell_2 + 2)\,;$$
$$\mathbf{assert}(!\,\ell_1 + !\,\ell_2 = 4)$$
$$\{\mathsf{True}\}$$

Example Program - Verified

$\{\mathsf{True}\}$

$\mathbf{let}\ \ell_1 = \mathbf{ref}\ 0\ \mathbf{in}$ // HT-LET, HT-ALLOC

$\{\ell_1 \mapsto 0\}$

$\mathbf{let}\ \ell_2 = \mathbf{ref}\ 0\ \mathbf{in}$

$(\ell_1 \leftarrow !\,\ell_1 + 2 \parallel \ell_2 \leftarrow !\,\ell_2 + 2)\,;$

$\mathbf{assert}(!\,\ell_1 + !\,\ell_2 = 4)$

$\{\mathsf{True}\}$

Example Program - Verified

$\{\mathsf{True}\}$

$\texttt{let}\ \ell_1 = \texttt{ref}\ 0\ \texttt{in}$ // HT-LET, HT-ALLOC

$\{\ell_1 \mapsto 0\}$

$\texttt{let}\ \ell_2 = \texttt{ref}\ 0\ \texttt{in}$ // HT-LET, HT-ALLOC, HT-FRAME

$\{\ell_1 \mapsto 0 * \ell_2 \mapsto 0\}$

$\left(\ell_1 \leftarrow !\,\ell_1 + 2 \;\middle\|\; \ell_2 \leftarrow !\,\ell_2 + 2\right);$

$\texttt{assert}(!\,\ell_1 + !\,\ell_2 = 4)$

$\{\mathsf{True}\}$

Example Program - Verified

$$
\begin{array}{ll}
\{\mathsf{True}\} & \\
\mathbf{let}\ \ell_1 = \mathbf{ref}\ 0\ \mathbf{in} & //\ \textsc{Ht-let}, \textsc{Ht-alloc} \\
\{\ell_1 \mapsto 0\} & \\
\mathbf{let}\ \ell_2 = \mathbf{ref}\ 0\ \mathbf{in} & //\ \textsc{Ht-let}, \textsc{Ht-alloc}, \textsc{Ht-frame} \\
\{\ell_1 \mapsto 0 * \ell_2 \mapsto 0\} & \\
\left(\begin{array}{l} \{\ell_1 \mapsto 0\} \\ \ell_1 \leftarrow\ !\,\ell_1 + 2 \end{array} \middle\| \begin{array}{l} \{\ell_2 \mapsto 0\} \\ \ell_2 \leftarrow\ !\,\ell_2 + 2 \end{array} \right); & //\ \textsc{Ht-seq}, \textsc{Ht-par} \\
\mathbf{assert}(!\,\ell_1 + !\,\ell_2 = 4) & \\
\{\mathsf{True}\} &
\end{array}
$$

Example Program - Verified

$\{\mathsf{True}\}$

$\texttt{let}\ \ell_1 = \texttt{ref}\ 0\ \texttt{in}$ // Ht-let, Ht-alloc

$\{\ell_1 \mapsto 0\}$

$\texttt{let}\ \ell_2 = \texttt{ref}\ 0\ \texttt{in}$ // Ht-let, Ht-alloc, Ht-frame

$\{\ell_1 \mapsto 0 * \ell_2 \mapsto 0\}$

$$\left(\begin{array}{l} \{\ell_1 \mapsto 0\} \\ \ell_1 \leftarrow !\,\ell_1 + 2 \\ \{\ell_1 \mapsto 2\} \end{array} \middle\| \begin{array}{l} \{\ell_2 \mapsto 0\} \\ \ell_2 \leftarrow !\,\ell_2 + 2 \\ \{\ell_2 \mapsto 2\} \end{array}\right);$$

// Ht-seq, Ht-par, Ht-load, Ht-store

$\texttt{assert}(!\,\ell_1 + !\,\ell_2 = 4)$

$\{\mathsf{True}\}$

Example Program - Verified

$$\{\mathsf{True}\}$$

$$\texttt{let}\ \ell_1 = \texttt{ref}\ 0\ \texttt{in} \qquad // \textsc{Ht-let}, \textsc{Ht-alloc}$$

$$\{\ell_1 \mapsto 0\}$$

$$\texttt{let}\ \ell_2 = \texttt{ref}\ 0\ \texttt{in} \qquad // \textsc{Ht-let}, \textsc{Ht-alloc}, \textsc{Ht-frame}$$

$$\{\ell_1 \mapsto 0 * \ell_2 \mapsto 0\}$$

$$\left(\begin{array}{l} \{\ell_1 \mapsto 0\} \\ \ell_1 \leftarrow !\,\ell_1 + 2 \\ \{\ell_1 \mapsto 2\} \end{array} \middle\| \begin{array}{l} \{\ell_2 \mapsto 0\} \\ \ell_2 \leftarrow !\,\ell_2 + 2 \\ \{\ell_2 \mapsto 2\} \end{array}\right); \qquad // \textsc{Ht-seq}, \textsc{Ht-par}, \textsc{Ht-load}, \textsc{Ht-store}$$

$$\{\ell_1 \mapsto 2 * \ell_2 \mapsto 2\}$$

$$\texttt{assert}(!\,\ell_1 + !\,\ell_2 = 4)$$

$$\{\mathsf{True}\}$$

Example Program - Verified

$$
\begin{array}{l}
\{\mathsf{True}\} \\
\texttt{let}\ \ell_1 = \texttt{ref}\ 0\ \texttt{in} \qquad // \textsc{Ht-let}, \textsc{Ht-alloc} \\
\{\ell_1 \mapsto 0\} \\
\texttt{let}\ \ell_2 = \texttt{ref}\ 0\ \texttt{in} \qquad // \textsc{Ht-let}, \textsc{Ht-alloc}, \textsc{Ht-frame} \\
\{\ell_1 \mapsto 0 * \ell_2 \mapsto 0\} \\
\left(\begin{array}{l} \{\ell_1 \mapsto 0\} \\ \ell_1 \leftarrow !\,\ell_1 + 2 \\ \{\ell_1 \mapsto 2\} \end{array} \middle\| \begin{array}{l} \{\ell_2 \mapsto 0\} \\ \ell_2 \leftarrow !\,\ell_2 + 2 \\ \{\ell_2 \mapsto 2\} \end{array} \right); \qquad // \textsc{Ht-seq}, \textsc{Ht-par}, \textsc{Ht-load}, \textsc{Ht-store} \\
\{\ell_1 \mapsto 2 * \ell_2 \mapsto 2\} \\
\texttt{assert}(!\,\ell_1 + !\,\ell_2 = 4) \qquad // \textsc{Ht-load}, \textsc{Ht-assert} \\
\{\mathsf{True}\}
\end{array}
$$

But What About Multiparty Channels?

Roundtrip program:

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

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

But What About Multiparty Channels?

Roundtrip program:

$$\textbf{let}\,(c_0, c_1, c_2) = \textbf{new\_chan}(3)\,\textbf{in}$$
$$\left(\begin{array}{l} \textbf{let}\,x = 40\,\textbf{in}\,c_0[1].\textbf{send}(x); \\ \textbf{assert}(c_0[2].\textbf{recv}() = x + 2) \end{array} \middle\| \begin{array}{l} \textbf{let}\,y = c_1[0].\textbf{recv}()\,\textbf{in} \\ c_1[2].\textbf{send}(y + 1) \end{array} \middle\| \begin{array}{l} \textbf{let}\,z = c_2[1].\textbf{recv}()\,\textbf{in} \\ c_2[0].\textbf{send}(z + 1) \end{array}\right)$$

Goal: Prove crash-freedom (safety) and verify asserts (functional correctness)
Sub-Goal: Hoare triples for multiparty channel primitives

Ht-new
$\{???\}\,\textbf{new\_chan}(n)\,\{???\}$

Ht-send
$\{???\}\,c[i].\textbf{send}(v)\,\{???\}$

Ht-recv
$\{???\}\,c[i].\textbf{recv}()\,\{???\}$

Tour of Multris

Multris

Channel endpoint ownership: $c \rightarrowtail p$

Multris

Channel endpoint ownership: $c \rightarrowtail p$

Protocols: $\mathbf{!}\,[i]\,(\vec{x}:\vec{\tau})\,\langle v\rangle.\,p \mid \mathbf{?}[i]\,(\vec{x}:\vec{\tau})\,\langle v\rangle.\,p \mid \textbf{end}$

Multris

Channel endpoint ownership: $c \rightarrowtail p$

Protocols: $\mathbf{!}\,[i]\,(\vec{x}:\vec{\tau})\,\langle v\rangle.\,p \mid \mathbf{?}[i]\,(\vec{x}:\vec{\tau})\,\langle v\rangle.\,p \mid \mathbf{end}$

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

Multris

Channel endpoint ownership: $c \rightarrowtail p$

Protocols: $\mathbf{!}\,[i]\,(\vec{x}:\vec{\tau})\,\langle v\rangle.\,p \mid \mathbf{?}[i]\,(\vec{x}:\vec{\tau})\,\langle v\rangle.\,p \mid \mathbf{end}$

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

Rules:

Ht-new
$$\{\textsc{consistent}\ \vec{p} * |\vec{p}| = n+1\}\ \texttt{new\_chan}(|\vec{p}|)\ \{(c_0,\ldots,c_n).\ c_0 \rightarrowtail \vec{p}_0 * \ldots * c_n \rightarrowtail \vec{p}_n\}$$

Multris

Channel endpoint ownership: $c \rightarrowtail p$

Protocols: $\mathbf{!}\,[i]\,(\vec{x}:\vec{\tau})\,\langle v\rangle.\,p \mid \mathbf{?}[i]\,(\vec{x}:\vec{\tau})\,\langle v\rangle.\,p \mid \mathbf{end}$

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

Rules:

HT-NEW

$$\{\textsc{consistent}\ \vec{p} * |\vec{p}| = n+1\}\ \mathbf{new\_chan}(|\vec{p}|)\ \{(c_0,\ldots,c_n).\ c_0 \rightarrowtail \vec{p}_0 * \ldots * c_n \rightarrowtail \vec{p}_n\}$$

HT-SEND

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

Multris

Channel endpoint ownership: $c \rightarrowtail p$

Protocols: $\mathbf{!}\,[i]\,(\vec{x}:\vec{\tau})\,\langle v\rangle.\,p \mid \mathbf{?}[i]\,(\vec{x}:\vec{\tau})\,\langle v\rangle.\,p \mid \mathbf{end}$

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

Rules:

HT-NEW

$$\{\textsc{consistent}\ \vec{p} * |\vec{p}| = n+1\}\ \mathtt{new\_chan}(|\vec{p}|)\ \{(c_0,\ldots,c_n).\ c_0 \rightarrowtail \vec{p}_0 * \ldots * c_n \rightarrowtail \vec{p}_n\}$$

HT-SEND

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

HT-RECV

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

Protocol Consistency

For any synchronised exchange from i to j, given the binders of i, we must:

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, given the binders of i, we must:

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.

$$\frac{(\forall i, j.\ \texttt{semantic\_dual}\ \vec{p}\ i\ j)}{\textsc{consistent}\ \vec{p}}*$$

$$\frac{\begin{array}{c}\vec{p}_i = \mathbf{!}\,[j]\,(\vec{x_1} : \vec{\tau_1})\,\langle v_1 \rangle.\, p_1 \twoheadrightarrow \vec{p}_j = \mathbf{?}[i]\,(\vec{x_2} : \vec{\tau_2})\,\langle v_2 \rangle.\, p_2 \twoheadrightarrow \\ \forall \vec{x_1} : \vec{\tau_1}.\ \exists \vec{x_2} : \vec{\tau_2}.\ v_1 = v_2 * \triangleright(\textsc{consistent}\ (\vec{p}[i := p_1][j := p_2]))\end{array}}{\texttt{semantic\_dual}\ \vec{p}\ i\ j}*$$

Protocol Consistency - Example

Protocol consistency example:

$$\vec{p}_0 := \,!\,[1]\,(x:\mathbb{Z})\,\langle x\rangle.\,?[2]\,\langle x+2\rangle.\,\textbf{end}$$
$$\vec{p}_1 := \,?[0]\,(y:\mathbb{Z})\,\langle y\rangle.\,!\,[2]\,\langle y+1\rangle.\,\textbf{end}$$
$$\vec{p}_2 := \,?[1]\,(z:\mathbb{Z})\,\langle z\rangle.\,!\,[0]\,\langle z+1\rangle.\,\textbf{end}$$

Protocol consistency:

$$\frac{(\forall i, j.\ \texttt{semantic\_dual}\ \vec{p}\ i\ j)}{\textsc{consistent}\ \vec{p}}*$$

$$\frac{\begin{array}{c}\vec{p}_i = \,!\,[j]\,(\vec{x_1}:\vec{\tau_1})\,\langle v_1\rangle.\,p_1 \twoheadrightarrow \vec{p}_j = \,?[i]\,(\vec{x_2}:\vec{\tau_2})\,\langle v_2\rangle.\,p_2 \twoheadrightarrow \\ \forall \vec{x_1}:\vec{\tau_1}.\ \exists \vec{x_2}:\vec{\tau_2}.\ v_1 = v_2 * \triangleright(\textsc{consistent}\ (\vec{p}[i := p_1][j := p_2]))\end{array}}{\texttt{semantic\_dual}\ \vec{p}\ i\ j}*$$

Roundtrip Example - Verified

Roundtrip program:

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

Protocols:

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

Roundtrip Example - Verified

Roundtrip program:

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

Protocols:

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

Verified Functional Correctness!

Roundtrip Reference Example

Roundtrip reference program:

$$\mathtt{let}\,(c_0, c_1, c_2) = \mathtt{new\_chan}(3)\,\mathtt{in}$$

$$\left(\begin{array}{l} \mathtt{let}\,x = 40\,\mathtt{in} \\ \mathtt{let}\,\ell = \mathtt{ref}\,x\,\mathtt{in} \\ c_0[1].\mathtt{send}(\ell); \\ c_0[2].\mathtt{recv}(); \\ \mathtt{assert}(!\,\ell = x + 2) \end{array} \middle\| \begin{array}{l} \mathtt{let}\,\ell = c_1[0].\mathtt{recv}()\,\mathtt{in} \\ \ell \leftarrow (!\,\ell + 1); \\ c_1[2].\mathtt{send}(\ell) \\ \\ \\ \end{array} \middle\| \begin{array}{l} \mathtt{let}\,\ell = c_2[0].\mathtt{recv}()\,\mathtt{in} \\ \ell \leftarrow (!\,\ell + 1); \\ c_2[0].\mathtt{send}(\ell) \\ \\ \\ \end{array}\right)$$

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

Multris with Resources

Protocols: $! \, [i] \, (\vec{x} : \vec{\tau}) \langle v \rangle \{P\}. \, p \mid ? \, [i] \, (\vec{x} : \vec{\tau}) \langle v \rangle \{P\}. \, p$

Example: $! \, [1] \, (\ell : \mathsf{Loc}, x : \mathbb{Z}) \, \langle \ell \rangle \{\ell \mapsto x\}. \, ?[2] \, \langle () \rangle \{\ell \mapsto (x+2)\}. \, \mathbf{end}$

Rules:

Ht-new

$$\{\textsc{consistent}\ \vec{p} * |\vec{p}| = n + 1\} \ \mathbf{new\_chan}(|\vec{p}|) \ \{(c_0, \ldots, c_n). \, c_0 \rightarrowtail \vec{p}_0 * \ldots * c_n \rightarrowtail \vec{p}_n\}$$

Ht-send

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

Ht-recv

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

Protocol Consistency with Resources

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

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.

$$\frac{(\forall i, j.\ \texttt{semantic\_dual}\ \vec{p}\ i\ j)}{\textsc{consistent}\ \vec{p}}*$$

$$\frac{\begin{array}{c}\vec{p}_i = \mathbf{!}\,[j]\,(\vec{x_1} : \vec{\tau_1})\langle v_1\rangle\{P_1\}.\,p_1 \twoheadrightarrow* \vec{p}_j = \mathbf{?}\,[i]\,(\vec{x_2} : \vec{\tau_2})\langle v_2\rangle\{P_2\}.\,p_2 \twoheadrightarrow* \\ \forall \vec{x_1} : \vec{\tau_1}.\ P_1 \twoheadrightarrow* \exists \vec{x_2} : \vec{\tau_2}.\ v_1 = v_2 * P_2 * \triangleright(\textsc{consistent}\ (\vec{p}[i := p_1][j := p_2]))\end{array}}{\texttt{semantic\_dual}\ \vec{p}\ i\ j}*$$

Protocol Consistency with Resources - Example

Protocol consistency example:

$$\vec{p}_0 := \,!\,[1]\,(\ell : \mathsf{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,?[2]\,\langle () \rangle \{\ell \mapsto (x+2)\}.\,\mathbf{end}$$
$$\vec{p}_1 := \,?[0]\,(\ell : \mathsf{Loc}, y : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto y\}.\,!\,[2]\,\langle \ell \rangle \{\ell \mapsto (y+1)\}.\,\mathbf{end}$$
$$\vec{p}_2 := \,?[1]\,(\ell : \mathsf{Loc}, z : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto z\}.\,!\,[0]\,\langle () \rangle \{\ell \mapsto (z+1)\}.\,\mathbf{end}$$

Protocol consistency:

$$\frac{(\forall i, j.\ \mathtt{semantic\_dual}\ \vec{p}\ i\ j)}{\textsc{consistent}\ \vec{p}}*$$

$$\frac{\vec{p}_i = \,!\,[j]\,(\vec{x_1} : \vec{\tau_1})\langle v_1 \rangle \{P_1\}.\,p_1 \twoheadrightarrow \vec{p}_j = \,?\,[i]\,(\vec{x_2} : \vec{\tau_2})\langle v_2 \rangle \{P_2\}.\,p_2 \twoheadrightarrow \quad \forall \vec{x_1} : \vec{\tau_1}.\ P_1 \twoheadrightarrow \exists \vec{x_2} : \vec{\tau_2}.\ v_1 = v_2 * P_2 * \triangleright(\textsc{consistent}\ (\vec{p}[i := p_1][j := p_2]))}{\mathtt{semantic\_dual}\ \vec{p}\ i\ j}*$$

Roundtrip Reference Example - Verified

Roundtrip reference program:

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

Protocols:

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

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

Protocol Consistency - Recursion

Protocols are contractive in the tail:

$$\mu rec.\, ![1]\,(\ell : \mathsf{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\, ?[2]\,\langle () \rangle \{\ell \mapsto (x + 2)\}.\, rec$$

Protocol Consistency - Recursion

Protocols are contractive in the tail:

$$\mu rec.\, !\,[1]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\, ?[2]\,\langle () \rangle \{\ell \mapsto (x+2)\}.\, rec$$

Protocol consistency example:

$$\begin{aligned}
\vec{p}_0 &= \mu rec.\, !\,[1]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\, ?[2]\,\langle () \rangle \{\ell \mapsto (x+2)\}.\, rec \\
\vec{p}_1 &= \mu rec.\, ?[0]\,(\ell : \mathrm{Loc}, y : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto y\}.\, !\,[2]\,\langle \ell \rangle \{\ell \mapsto (y+1)\}.\, rec \\
\vec{p}_2 &= \mu rec.\, ?[1]\,(\ell : \mathrm{Loc}, z : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto z\}.\, !\,[0]\,\langle () \rangle \{\ell \mapsto (z+1)\}.\, rec
\end{aligned}$$

Recursion via Löb induction ($\triangleright$)

$$\frac{\begin{array}{c}\vec{p}_i = !\,[j]\,(\vec{x_1} : \vec{\tau_1})\,\langle v_1 \rangle \{P_1\}.\, p_1 \twoheadrightarrow \vec{p}_j = ?[i]\,(\vec{x_2} : \vec{\tau_2})\,\langle v_2 \rangle \{P_2\}.\, p_2 \twoheadrightarrow \\ \forall \vec{x_1} : \vec{\tau_1}.\, P_1 \twoheadrightarrow \exists \vec{x_2} : \vec{\tau_2}.\, v_1 = v_2 * P_2 * \triangleright(\textsc{consistent}\,(\vec{p}[i := p_1][j := p_2]))\end{array}}{\texttt{semantic\_dual}\,\vec{p}\,i\,j} *$$

Protocol Consistency - Framing

Consider the replacement of process 1 with a forwarder:

$$\texttt{let}\; v = c_1[0].\texttt{recv}()\; \texttt{in}\; c_1[1].\texttt{send}(v)$$

Protocol Consistency - Framing

Consider the replacement of process 1 with a forwarder:

$$\texttt{let}\, v = c_1[0].\texttt{recv}()\,\texttt{in}\, c_1[1].\texttt{send}(v)$$

Protocol consistency example:

$$\begin{aligned}
\vec{p}_0 &= \mu rec.\,!\,[1]\,(\ell : \mathsf{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,?[2]\,\langle () \rangle \{\ell \mapsto (x+1)\}.\,rec \\
\vec{p}_1 &= \mu rec.\,?[0]\,(\ell : \mathsf{Loc}, y : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto y\}.\,!\,[2]\,\langle \ell \rangle \{\ell \mapsto y\}.\,rec \\
\vec{p}_2 &= \mu rec.\,?[1]\,(\ell : \mathsf{Loc}, z : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto z\}.\,!\,[0]\,\langle () \rangle \{\ell \mapsto (z+1)\}.\,rec
\end{aligned}$$

Protocol Consistency - Framing

Consider the replacement of process 1 with a forwarder:

$$\texttt{let}\ v = c_1[0].\texttt{recv}()\ \texttt{in}\ c_1[1].\texttt{send}(v)$$

Protocol consistency example:

$$\begin{aligned}
\vec{p}_0 &= \mu rec.\, ![1]\,(\ell : \mathsf{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\, ?[2]\,\langle () \rangle \{\ell \mapsto (x+1)\}.\, rec \\
\vec{p}_1 &= \mu rec.\, ?[0]\,(v : \mathsf{Val})\,\langle v \rangle.\, ![2]\,\langle v \rangle.\, rec \\
\vec{p}_2 &= \mu rec.\, ?[1]\,(\ell : \mathsf{Loc}, z : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto z\}.\, ![0]\,\langle () \rangle \{\ell \mapsto (z+1)\}.\, rec
\end{aligned}$$

Protocol Consistency - Framing

Consider the replacement of process 1 with a forwarder:

$$\texttt{let}\ v = c_1[0].\texttt{recv}()\ \texttt{in}\ c_1[1].\texttt{send}(v)$$

Protocol consistency example:

$$\begin{aligned}
\vec{p}_0 &= \mu rec.\,!\,[1]\,(\ell : \mathrm{Loc}, x : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto x\}.\,?[2]\,\langle () \rangle \{\ell \mapsto (x+1)\}.\,rec \\
\vec{p}_1 &= \mu rec.\,?[0]\,(v : \mathrm{Val})\,\langle v \rangle.\,!\,[2]\,\langle v \rangle.\,rec \\
\vec{p}_2 &= \mu rec.\,?[1]\,(\ell : \mathrm{Loc}, z : \mathbb{Z})\,\langle \ell \rangle \{\ell \mapsto z\}.\,!\,[0]\,\langle () \rangle \{\ell \mapsto (z+1)\}.\,rec
\end{aligned}$$

Protocol consistency owns resources while in transit:

$$\frac{\begin{array}{c}\vec{p}_i = !\,[j]\,(\vec{x_1} : \vec{\tau_1})\,\langle v_1 \rangle \{P_1\}.\,p_1 \twoheadrightarrow \vec{p}_j = ?[i]\,(\vec{x_2} : \vec{\tau_2})\,\langle v_2 \rangle \{P_2\}.\,p_2 \twoheadrightarrow \\ \forall \vec{x_1} : \vec{\tau_1}.\,P_1 \twoheadrightarrow \exists \vec{x_2} : \vec{\tau_2}.\,v_1 = v_2 * P_2 * \triangleright(\textsc{consistent}\,(\vec{p}[i := p_1][j := p_2]))\end{array}}{\texttt{semantic\_dual}\ \vec{p}\ i\ j}*$$

Protocol Consistency - Branching

Consider the extension of process 1 with a rerouter:

$$\mathtt{let}\,(v, b) = c_1[0].\mathtt{recv}()\;\mathtt{in}\;c_1[\mathtt{if}\;b\;\mathtt{then}\;2\;\mathtt{else}\;3].\mathtt{send}(v)$$

Protocol Consistency - Branching

Consider the extension of process 1 with a rerouter:

$$\texttt{let}\,(v,b) = c_1[0].\texttt{recv}()\,\texttt{in}\,c_1[\texttt{if}\,b\,\texttt{then}\,2\,\texttt{else}\,3].\texttt{send}(v)$$

Protocol consistency example:

$$\begin{aligned}
\vec{p}_0 &= \mu rec.\,!\,[1]\,(\ell : \mathrm{Loc}, x : \mathbb{Z}, b : \mathbb{B})\,\langle(\ell, b)\rangle\{\ell \mapsto x\}. \\
&\qquad ?[\texttt{if}\,b\,\texttt{then}\,2\,\texttt{else}\,3]\,\langle()\rangle\{\ell \mapsto (x+1)\}.\,rec \\
\vec{p}_1 &= \mu rec.\,?[0]\,(v : \mathrm{Val}, b : \mathbb{B})\,\langle(v, b)\rangle.\,!\,[\texttt{if}\,b\,\texttt{then}\,2\,\texttt{else}\,3]\,\langle v\rangle.\,rec \\
\vec{p}_2, \vec{p}_3 &= \mu rec.\,?[1]\,(\ell : \mathrm{Loc}, z : \mathbb{Z})\,\langle\ell\rangle\{\ell \mapsto z\}.\,!\,[0]\,\langle()\rangle\{\ell \mapsto (z+1)\}.\,rec
\end{aligned}$$

We can do case analysis on the binders:

$$\frac{\begin{array}{c}\vec{p}_i = !\,[j]\,(\vec{x_1} : \vec{\tau_1})\,\langle v_1\rangle\{P_1\}.\,p_1 \twoheadrightarrow \vec{p}_j = ?[i]\,(\vec{x_2} : \vec{\tau_2})\,\langle v_2\rangle\{P_2\}.\,p_2 \twoheadrightarrow \\ \forall \vec{x_1} : \vec{\tau_1}.\,P_1 \twoheadrightarrow \exists \vec{x_2} : \vec{\tau_2}.\,v_1 = v_2 * P_2 * \triangleright(\textsc{consistent}\,(\vec{p}[i := p_1][j := p_2]))\end{array}}{\texttt{semantic\_dual}\,\vec{p}\,i\,j}*$$

Language Parametricity of Multris

Multris Ghost Theory

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

PROTO-ALLOC

$$\frac{\textsc{consistent}\ \vec{p}}{\Rrightarrow \exists \chi.\ \mathsf{prot\_ctx}\ \chi\ |\vec{p}| * \mathop{\Huge *}_{i \mapsto p \in \vec{p}} \mathsf{prot\_own}\ \chi\ i\ p}$$

PROTO-VALID

$$\frac{\mathsf{prot\_ctx}\ \chi\ n \qquad \mathsf{prot\_own}\ \chi\ i\ p}{i < n}$$

PROTO-STEP

$$\frac{\begin{array}{c}\mathsf{prot\_ctx}\ \chi\ n \qquad P_1[\vec{t_1}/\vec{x_1}] \\ \mathsf{prot\_own}\ \chi\ i\ (!\,[j]\,(\vec{x_1} : \vec{\tau_1})\,\langle v_1\rangle\{P_1\}.\,p_1) \qquad \mathsf{prot\_own}\ \chi\ j\ (?[i]\,(\vec{x_2} : \vec{\tau_2})\,\langle v_2\rangle\{P_2\}.\,p_2)\end{array}}{\begin{array}{c}\Rrightarrow \triangleright \exists (\vec{t_2} : \vec{\tau_2}).\ \mathsf{prot\_ctx}\ \chi\ * \mathsf{prot\_own}\ \chi\ i\ (p_1[\vec{t_1}/\vec{x_1}]) * \mathsf{prot\_own}\ \chi\ j\ (p_2[\vec{t_2}/\vec{x_2}]) * \\ (v_1[\vec{t_1}/\vec{x_1}]) = (v_2[\vec{t_2}/\vec{x_2}]) * P_2[\vec{t_2}/\vec{x_2}]\end{array}}$$

One can then define language-specific $c \rightarrowtail p$ and prove Hoare triple rules (such as HT-SEND, HT-RECV, and HT-NEW) for a given language using the ghost theory

Conclusion and Future Work

Conclusion

Dependent multiparty protocols are non-trivial to prove sound

- Mismatched dependencies (quantifiers) makes syntatic analysis difficult
- Fullfillment of received resources is tricky

Concurrent separation logic is a good fit for multiparty protocols

- Quantifier scopes enable inherent tracking of dependencies
- Separation logic enables framing of resources

Mechanisation yields crucial level of automation

- Imperative for non-trivial multiparty protocol consistency proofs

Future Work

Additional features

- Asynchronous communication/subprotocols
- Mixed choice

Semantic Multiparty Session Type System

- Investigate correspondences with syntactic protocol consistency

Better methodology for proving protocol consistency

- Abstraction and Modularity via separation logic

Deadlock freedom guarantees

- Leverage connectivity graphs for multiparty communication

Multris for distributed systems

- Leverage the Aneris separation logic

Future Work

Additional features

- Asynchronous communication/subprotocols
- Mixed choice

Semantic Multiparty Session Type System

- Investigate correspondences with syntactic protocol consistency

Better methodology for proving protocol consistency

- Abstraction and Modularity via separation logic

Deadlock freedom guarantees

- Leverage connectivity graphs for multiparty communication

Multris for distributed systems

- Leverage the Aneris separation logic

And much more!

$$
\begin{aligned}
&\mathbf{!}\,[1]\,\langle\text{"Thank you"}\rangle\{\texttt{MultrisOverview}\}.\\
&\mu rec.\,\mathbf{?}[1]\,(q : \texttt{Question i})\,\langle q\rangle\{\texttt{AboutMultris}\ q\}.\\
&\qquad\mathbf{!}\,[i]\,(a : \texttt{Answer})\,\langle a\rangle\{\texttt{Insightful}\ q\ a\}.\,rec
\end{aligned}
$$