NUS National University of Singapore

School of Computing

A Session Logic for Relaxed Communication Protocols

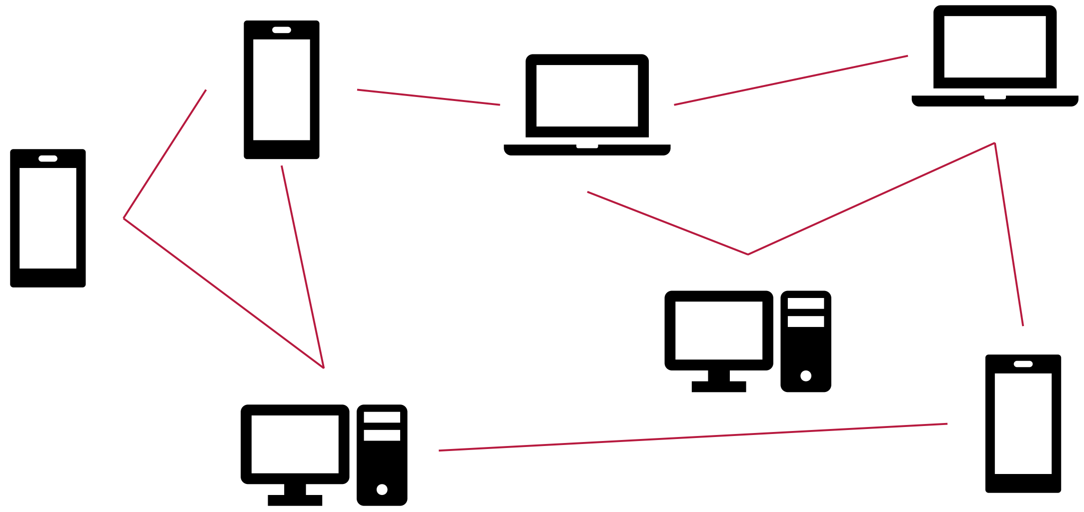
Andreea Costea

Department of Computer Science

Advisor: A/P Wei-Ngan Chin

Thesis Defense

6th December 2017



Systems development life cycle:

Design → Specify → ? → Implement → Validate → Operate & Maintain

(communication protocols)

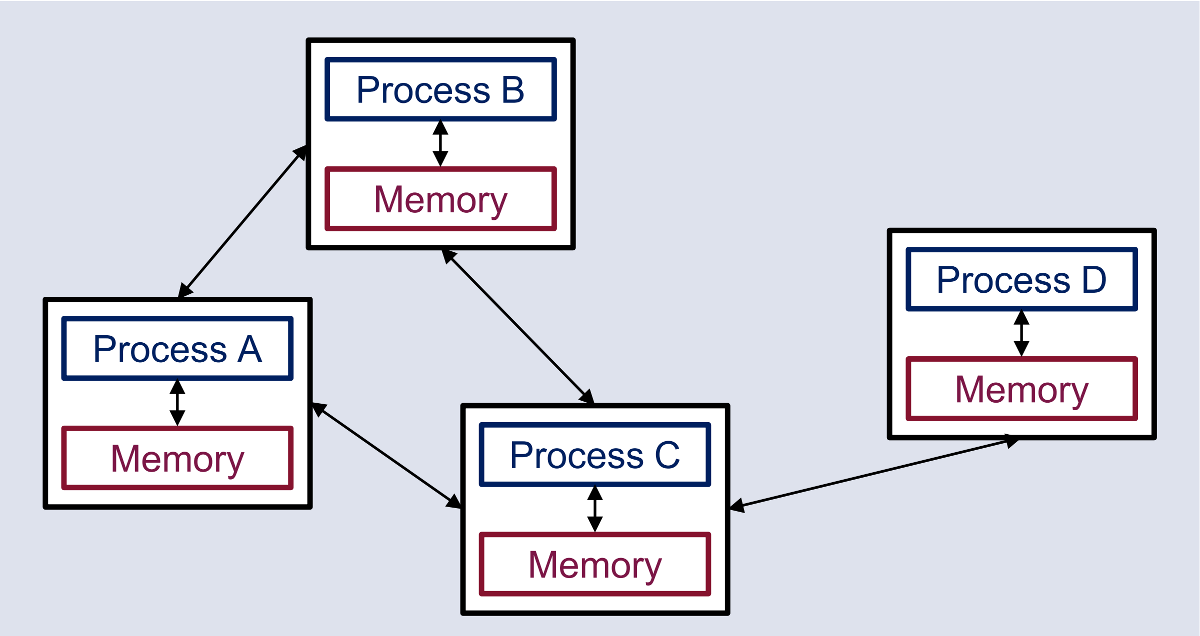
(different dev teams)

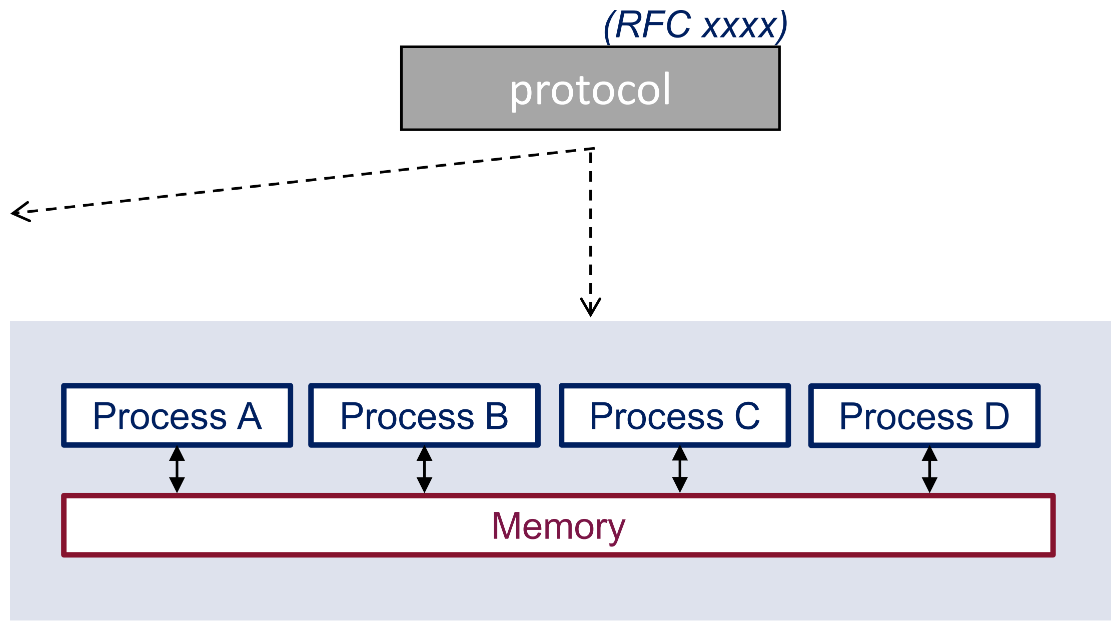
“A communication protocol defines the format and the order of messages exchanged between two or more communicating entities”. [Kurose and Ross]

Example of protocols: payment systems, smart contracts, NFS, Linux boot protocol, FTP, etc

Q1: How to ensure that a protocol is correctly implemented?

Implementation of Protocols: loosely or tightly coupled





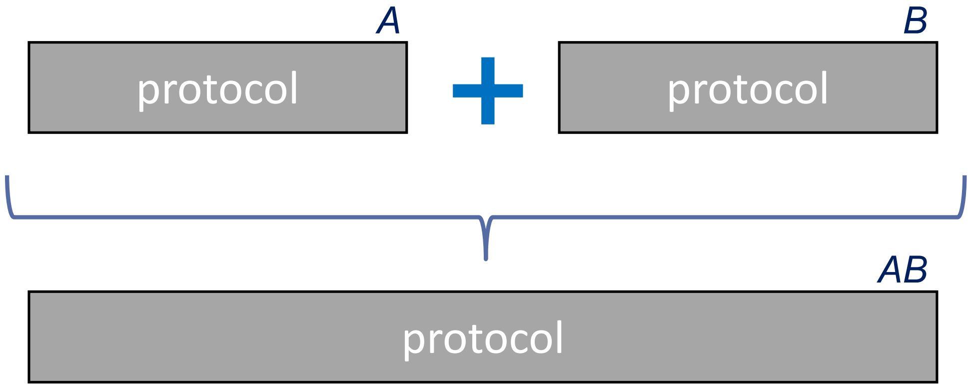
Design → Specify → Implement ? *(network of dev teams)* → Validate → Operate & Maintain

Writing software is **error-prone.**

Writing **communication-centered** software even more so!

Q2: How to ensure that implementations are safe?

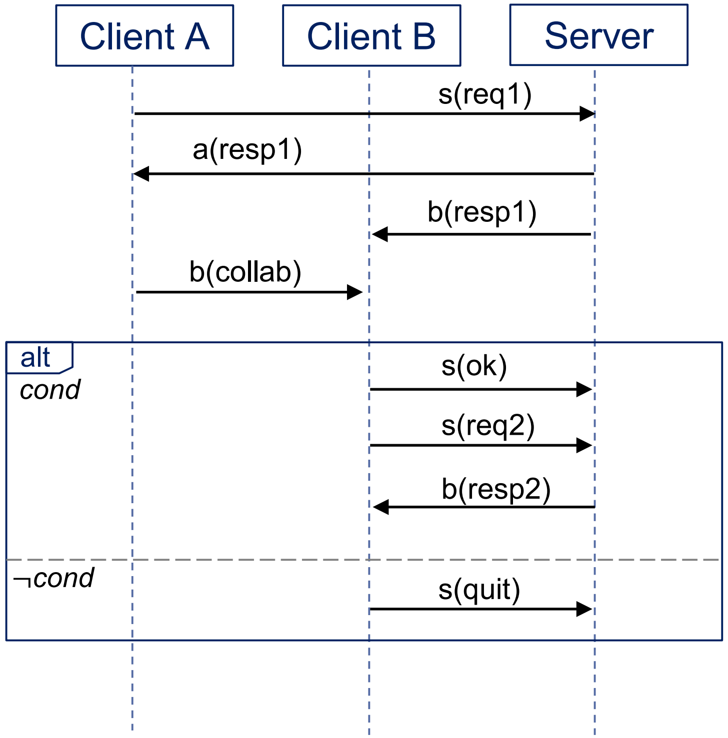
Compatibility of Protocols



Q3: How to ensure that protocols are safely composed?

A Telling Example

Collaborative Client – Server*



Protocol Elements:

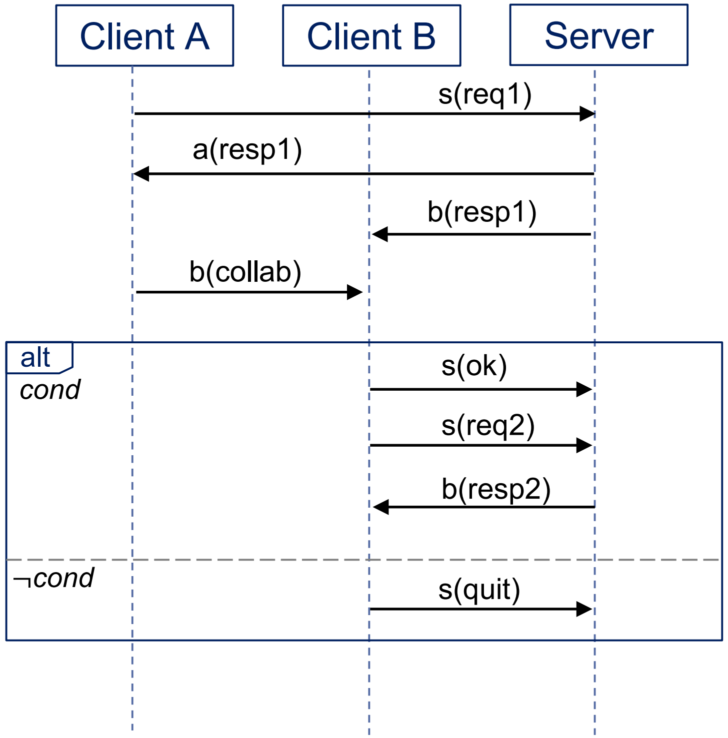
- communicating entities (parties): Client A, Client B, Server
- messages: req, resp, collab, ok, quit
- direction and order of transmission
- channel: a, b, s
- conditioned communication: cond

Communication Model:

- asynchronous communication
- FIFO mailbox channels

*Usages: Two Buyers - One Seller Protocol [Honda et al., 2008], Intel CS for WebRTC, Hybrid client-server for 3D design [Desprat et al. 2015], Collaborative Remote Experimentation [Callaghan et al. 2014], etc.

Collaborative Client – Server



Buyer A

```
int price,share;
String book;
…
send(s, book);
price = receive(a);
share = foo(price);
send(b, share);
```

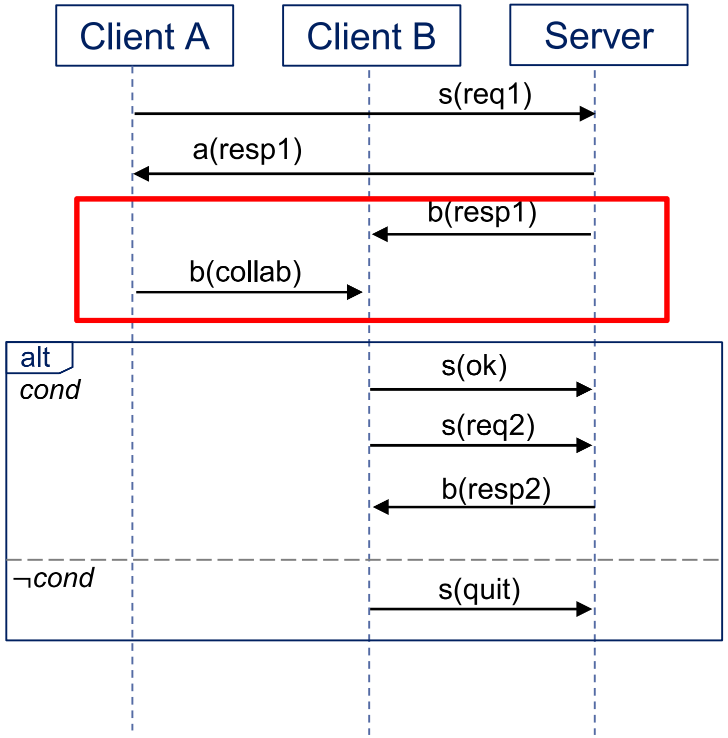
Buyer B

```
int price,clb;

…
price = receive(b);
clb = receive(b);
if(cond){
  send(s, ok);
  send(s, addr);
  … = receive(s);
}else{
 send(s, quit);
}
```

Seller

```
int id, val;

…
id = receive(s);
val = goo(id);
send(a,val);
send(b,val);
ans = receive(s);
if (s==ok){
  … = receive(s);
  send(b,…);
}
```

Collaborative Client – Server



Buyer A

```
int price,share;
String book;
…
send(s, book);
price = receive(a);
share = foo(price);
send(b, share);
```

Buyer B

```
int price,clb;

…
price = receive(b);
clb = receive(b);
if(cond){
   send(s, ok);
   send(s, addr);
   … = receive(s);
}else{
  send(s, quit);
}
```

Seller

```
int id, val;

…
id = receive(s);
val = goo(id);
send(a,val);
send(b,val);
ans = receive(s);
if (s==ok){
   … = receive(s);
   send(b,…);
}
```

🚫 Unsafe type manipulation

🚫 Race: non-linear usage channel b

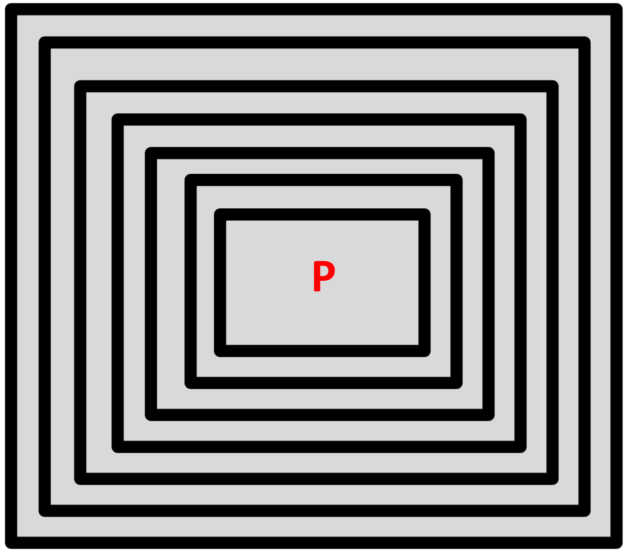
How to Deal with Software Bugs?

Testing

Is it good enough?

"Testing only shows the presence of bugs, not their absence."
Edsger W. Dijkstra

HW & SW Mitigation Solutions



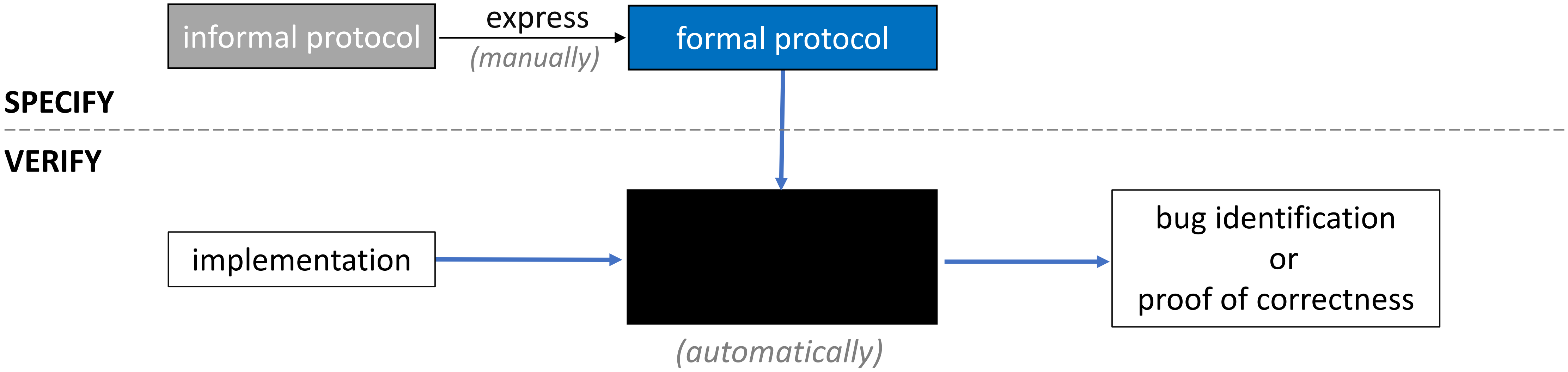
The Programming Language Approach

Given a notion of computation,

design a notation to express this computation

together with reasoning tools for that notation.

A Language-Based Approach to Formalizing Protocols



Thesis:

Language support makes it possible:

- to **specify** communication protocols, and then
- to **verify** (automatically) that an implementation conforms to the given protocol in a safe way.

Outline Of The Talk

1. Related Work
2. Session Logic
   A. Specification Language
   B. Identify Race Conditions
   C. Relaxed Protocols
   D. Modular Protocols
3. Communication Verification
4. Conclusion and Future Work

1. Related Work

2. Session Logic

   A. Specification Language
   B. Identify Race Conditions
   C. Relaxed Protocols
   D. Modular Protocols

3. Communication Verification

4. Conclusion and Future Work

State of the Art (1)

Shared Channel	
≥2 participants	
Linear implicitly synchronized transmissions.	**Non-linear** transmission with no causal relations.

Binary Session Types [HONDA et al. @ESOP'98]

- Subtyping [GAY & HOLE @AI'05]
- Sessions as effects [ORCHARD & YOSHIDA et al. @POPL'16]
- Embedding to Haskell [NEUBAUER & THIEMANN @PADL'04], multi-threaded ML [VASCONCELOS et al., @TCS'06], F# [Corin et al. @CFS'07], Java [Ciancaglini et al. ECOOP'06], etc

Multiparty Session Types [HONDA et al. @POPL'08]

- Progress – ***disallow shared channels*** [BETTINI et al. @CONCUR'08, COPPO et al. @MSCS'16]
- Linearity – ***shared channels are a must*** [CAIRES & PFENNING @CONCUR'10, GIUNTI & VASCONCELOS @MSCS'14, SCALAS et al. @ECOOP'17]
- Adding contracts [BOCCHI et al. @CONCUR'10], synthesize deadlock-free choreographies [CARBONE & MONTENSI @POPL'13], dynamic multirole [DENIELOU & YOSHIDA @POPL'11], nested sessions [DEMANGEON & HONDA @CONCUR12], safety for Go programs [YOSHIDA et al @POPL'17]
- Correspondence with linear logic [CAIRES & PFENNING @CONCUR'10, CAIRES et al. @MSCS'12, CARBONE et al. @CONCUR'15, CARBONE et al. @CONCUR'16, CARBONE et al. @AI'17]

State of the Art (2)

Program Logics and Tools For Concurrency

- Concurrent Separation Logic [O'HEARN @CONCUR'04]
- iCAP [SVENDSEN and BIRKEDAL @ESOP'14]
- locks [DODDS et al. @POPL'11], barriers [HOBOR & GHERGHINA, ESOP'18], higher-order functions [NANEVSKI et al. @ESOP'14],
- SmallfootRG [VAFEIADIS et al., CONCUR'07], Iris [JUNG et al. @POPL'15], VeriFast [JACOBS et al. @NFM'11], Infer @Facebook, SLAyer [Berdine @CAV'11]

Verification of Protocols

- Separation in time + Separation in space [HOARE and O'HEARN @TCS'08]
- CSL for copyless message passing [VILLARD et al. @APLAS'09]
- Chalice: message passing + locking [LEINO et al. @ESOP'10]
- IronFleet: proves safety and liveness [HAWBLITZEL et al. @SOSP'15]
- Verdi: vertical composition of protocols [WILCOX et al. @PLDI'15]
- DISEL: mechanized proofs for consensus protocols [SERGEY et al. @POPL'18]

1. Related Work

2. Session Logic

   A. Specification Language
   B. Identify Race Conditions
   C. Relaxed Protocols
   D. Modular Protocols

3. Communication Verification

4. Conclusion and Future Work

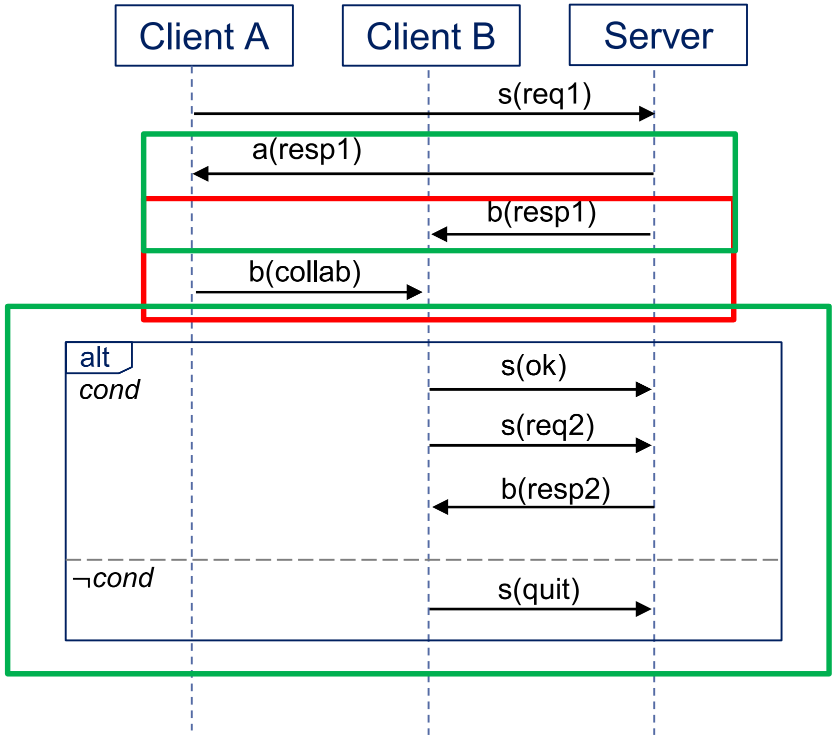
2A. Specification Language

Specification Language for Protocols

$$
\begin{array}{lll}
\textit{Global protocol} & G & ::= \\
\textit{Single transmission} & & \quad \mathtt{S} \xrightarrow{i} \mathtt{R} : \mathtt{c}\langle \mathtt{v} \cdot \Delta \rangle \\
\textit{Concurrency} & & \mid G * G \\
\textit{Choice} & & \mid G \vee G \\
\textit{Sequencing} & & \mid G \,;\, G \\
\textit{Inaction} & & \mid \mathtt{emp}
\end{array}
$$

(Parties) $\mathtt{P}, \mathtt{S}, \mathtt{R} \in \mathcal{R}\mathtt{ole}$ *(Channels)* $\mathtt{c} \in \mathcal{C}\mathtt{han}$ *(Messages)* $\mathtt{v} \cdot \Delta$ *(Labels)* $i \in \mathtt{Nat}$

Collaborative Client – Server (revisited)



$$
\begin{aligned}
G_{\mathrm{ABS}} \triangleq\ & \mathtt{A} \xrightarrow{1} \mathtt{S} : \mathtt{s}\langle \mathtt{v} \cdot \mathtt{v} : \mathtt{String}\rangle\,; \\
& \left(\mathtt{S} \xrightarrow{2} \mathtt{A} : \mathtt{a}\langle \mathtt{v} \cdot \mathtt{v} > 0\rangle * \mathtt{S} \xrightarrow{3} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} > 0\rangle\right); \mathtt{A} \xrightarrow{4} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} \geq 0\rangle\,; \\
& \left(\mathtt{B} \xrightarrow{5} \mathtt{S} : \mathtt{s}\langle \mathtt{ok}\rangle\,; \mathtt{B} \xrightarrow{6} \mathtt{S} : \mathtt{s}\langle \mathtt{v} \cdot \mathtt{Addr}(\mathtt{v})\rangle\,; \mathtt{S} \xrightarrow{7} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{Date}(\mathtt{v})\rangle \right. \\
& \left. \vee\, \mathtt{B} \xrightarrow{8} \mathtt{S} : \mathtt{s}\langle \mathtt{quit}\rangle\right).
\end{aligned}
$$

Different from session types:

1. Messages are described by *logical formulae*.
2. *Concurrent/arbitrary-ordered* transmissions.
3. Uniform treatment of internal/external choice via *disjunction*.

Take – away 1: TYPE SYSTEMS -> LOGIC

2B. Race-Free Conditions

Race Handling

$$\boxed{(\mathtt{S} \xrightarrow{2} \mathtt{A} : \mathtt{a}\langle \mathtt{v} \cdot \mathtt{v} > 0 \rangle \ * \ \mathtt{S} \xrightarrow{3} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} > 0 \rangle) \, ; \, \mathtt{A} \xrightarrow{4} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} \geq 0 \rangle}$$

Race Handling

$$(\mathtt{S} \xrightarrow{2} \mathtt{A} : \mathtt{a}\langle \mathtt{v} \cdot \mathtt{v} > 0 \rangle \ast \mathtt{S} \xrightarrow{3} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} > 0 \rangle) \,;\, \mathtt{A} \xrightarrow{4} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} \geq 0 \rangle$$

	Buyer A	Buyer B	Seller

(1)	…
	 `send(b, share);`
		…
		 `price = receive(b);`
		 `clb = receive(b);`
			…
			 `send(b,val);`

(2)	…
	 `wait(cnd);`
	 `send(b, share);`
		…
		 `price = receive(b);`
		 `notifyAll(cnd);`
		 `clb = receive(b);`
			…
			 `send(b,val);`

Current approaches for protocol formalization declare non-linear protocols as UNSAFE!

Our goal: *relax* the tag of “UNSAFE” non-linear protocols, by enforcing safety at the program code level.

Race Handling

$$(\mathtt{S} \xrightarrow{2} \mathtt{A} : \mathtt{a}\langle \mathtt{v} \cdot \mathtt{v} > 0\rangle \ast \mathtt{S} \xrightarrow{3} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} > 0\rangle)\,;\, \mathtt{A} \xrightarrow{4} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} \geq 0\rangle$$

	Buyer A	Buyer B	Seller

(1)	…
	`send(b, share);`
		…
		`price = receive(b);`
		`clb = receive(b);`
			…
			`send(b,val);`

(2)	…
	`wait(cnd);`
	`send(b, share);`
		…
		`price = receive(b);`
		`notifyAll(cnd);`
		`clb = receive(b);`
			…
			`send(b,val);`

Introduce a proof obligation on event ordering to prove that
$\mathrm{S}^{(3)}$ ***happens-before*** $\mathrm{A}^{(4)}$

Race Handling

$$\mathtt{S}_1 \xrightarrow{\mathtt{i}_1} \mathtt{R}_1 : \mathtt{c}\langle \Delta_1 \rangle ; \mathtt{S}_2 \xrightarrow{\mathtt{i}_2} \mathtt{R}_2 : \mathtt{c}\langle \Delta_2 \rangle$$

To ensure race-freedom on c, prove that:

S_1 ***happens-before*** S_2

and

R_1 ***happens-before*** R_2

Race Handling

$$\mathtt{S}_1 \xrightarrow{\mathtt{i}_1} \mathtt{R}_1 : \mathtt{c}\langle \Delta_1 \rangle ; \mathtt{S}_2 \xrightarrow{\mathtt{i}_2} \mathtt{R}_2 : \mathtt{c}\langle \Delta_2 \rangle$$

To ensure race-freedom on c, prove that:

(HB between transmissions)

$$\mathtt{S}_1^{(i_1)} \prec_{\mathtt{HB}} \mathtt{S}_2^{(i_2)} \;\wedge\; \mathtt{R}_1^{(i_1)} \prec_{\mathtt{HB}} \mathtt{R}_2^{(i_2)} \qquad \Leftrightarrow \qquad i_1 \prec_{\mathtt{HB}} i_2$$

Properties of the HB relation

1. **Transitive:** $\forall\, \mathtt{E}_1, \mathtt{E}_2, \mathtt{E}_3 \cdot \mathtt{E}_1 \prec_{\mathtt{HB}} \mathtt{E}_2 \wedge \mathtt{E}_2 \prec_{\mathtt{HB}} \mathtt{E}_3 \Rightarrow \mathtt{E}_1 \prec_{\mathtt{HB}} \mathtt{E}_3.$
2. **Irreflexive:** $\forall\, \mathtt{E}_1, \mathtt{E}_2 \cdot \mathtt{E}_1 \prec_{\mathtt{HB}} \mathtt{E}_2 \Rightarrow \mathtt{label}(\mathtt{E}_1) \neq \mathtt{label}(\mathtt{E}_2)$
3. **Asymmetric:** $\forall\, \mathtt{E}_1, \mathtt{E}_2 \cdot \mathtt{E}_1 \prec_{\mathtt{HB}} \mathtt{E}_2 \Rightarrow \neg(\mathtt{E}_2 \prec_{\mathtt{HB}} \mathtt{E}_1)$

Orderings Constraint System

Denotes a "*communicates-before*" relation:

$$\mathtt{S} \xrightarrow{i} \mathtt{R} : \mathtt{c}\langle \mathtt{v} \cdot \Delta \rangle \quad \Rightarrow \quad \mathtt{S}^{(i)} \prec_{\mathtt{CB}} \mathtt{R}^{(i)}$$

$$\begin{array}{lll}
\textit{Send/Recv Event} & \mathtt{E} & ::= \mathtt{P}^{(i)} \\
\textit{Ordering Constraints} & \vartheta & ::= \boxed{\mathtt{E} \prec_{\mathtt{CB}} \mathtt{E}} \mid \mathtt{E} \prec_{\mathtt{HB}} \mathtt{E} \\
\textit{Race-Free Assertions} & \Psi & ::= \mathtt{E} \mid \neg(\mathtt{E}) \mid \vartheta \mid \Psi \wedge \Psi \mid \mathtt{E} \Rightarrow \Psi
\end{array}$$

(a) Syntax of the ordering-constraints language

$$\begin{array}{lll}
\mathtt{E}_1 \prec_{\mathtt{HB}} \mathtt{E}_2 \wedge \mathtt{E}_2 \prec_{\mathtt{HB}} \mathtt{E}_3 & \Rightarrow \mathtt{E}_1 \prec_{\mathtt{HB}} \mathtt{E}_3 & \text{[HB-HB]} \\
\mathtt{E}_1 \prec_{\mathtt{CB}} \mathtt{E}_2 \wedge \mathtt{E}_2 \prec_{\mathtt{HB}} \mathtt{E}_3 & \Rightarrow \mathtt{E}_1 \prec_{\mathtt{HB}} \mathtt{E}_3 & \text{[CB-HB]}
\end{array}$$

(b) Constraint propagation rule

$$\begin{array}{llll}
\Pi \models \mathtt{E} & \text{iff } \mathtt{E} \in \Pi & \Pi \models \mathtt{E} \Rightarrow \Psi & \text{iff } \neg(\Pi \models \mathtt{E}) \text{ or } \Pi \models \Psi \\
\Pi \models \neg(\mathtt{E}) & \text{iff } \mathtt{E} \notin \Pi & \Pi \models \Psi_1 \wedge \Psi_2 & \text{iff } \Pi \models \Psi_1 \text{ and } \Pi \models \Psi_2
\end{array}$$

$$\Pi \models \mathtt{E}_1 \prec_{\mathtt{HB}} \mathtt{E}_2 \quad \text{iff } \Big(\bigwedge_{\Psi \in \Pi} \Psi \Big) \Rightarrow^* \mathtt{E}_1 \prec_{\mathtt{HB}} \mathtt{E}_2$$

(c) Semantics of race-free assertions, where Π is a set of events and ordering constraints.

Take – away 2: TEMPORAL ORDERING

Race Formalization

Definition: Race Relation

A race relation $\texttt{RACE} \subseteq \mathit{Transmission} \times \mathit{Transmission}$ is defined as follows:

$$\{(i_1, i_2) \mid i_1, i_2 \in G \cdot i_1 \neq i_2 \wedge (\texttt{Adj}^+(i_1, i_2) \Rightarrow \neg(i_1 \prec_{\texttt{HB}} i_2))\}.$$

Definition: Race-free Relation

A race relation $\texttt{RF} \subseteq \mathit{Transmission} \times \mathit{Transmission}$ is defined as follows:

$$\{(i_1, i_2) \mid i_1, i_2 \in G \cdot i_1 \neq i_2 \wedge (\texttt{Adj}^+(i_1, i_2) \Rightarrow i_1 \prec_{\texttt{HB}} i_2)\}.$$

Race Formalization (cont.)

Definition: Race-free Protocol

A protocol G is race-free, denoted by $\mathtt{RF}(G)$, if all the linked transmissions are race-free:

$$\forall i_1, i_2 \in G \cdot \mathtt{Adj}^+(i_1, i_2) \Rightarrow \mathtt{RF}(i_1, i_2).$$

Theorem: Race-free Protocol

A protocol G is race-free if and only if all the adjacent transmissions are race-free:

$$(\forall i_1, i_2 \in G \cdot \mathtt{Adj}(i_1, i_2) \Rightarrow \mathtt{RF}(i_1, i_2)) \Leftrightarrow \mathtt{RF}(G).$$

Take – away 3: RACE-FREE PROTOCOLS

2C. Relaxed Protocols

Race Handling

$$(\mathtt{S} \xrightarrow{2} \mathtt{A} : \mathtt{a}\langle \mathtt{v} \cdot \mathtt{v} > 0 \rangle \ast \mathtt{S} \xrightarrow{3} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} > 0 \rangle) \,;\, \mathtt{A} \xrightarrow{4} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} \geq 0 \rangle$$

	Buyer A	Buyer B	Seller

(1)	…
	`send(b, share);`
		…
		`price = receive(b);`
		`clb = receive(b);`
			…
			`send(b,val);`

(2)	…
	`wait(cnd);`
	`send(b, share);`
		…
		`price = receive(b);`
		`notifyAll(cnd);`
		`clb = receive(b);`
			…
			`send(b,val);`

Current approaches for protocol formalization declare non-linear protocols as UNSAFE!

Our goal: *relax* the tag of “UNSAFE” non-linear protocols, by enforcing safety at the program code level.

Specification Language for Relaxed Protocols

$$
\begin{array}{lrl}
\textit{Global protocol} & G & ::= \\
\textit{Single transmission} & & \mathtt{S} \xrightarrow{i} \mathtt{R} : \mathtt{c}\langle \mathtt{v} \cdot \Delta \rangle \\
\textit{Concurrency} & & \mid\ G * G \\
\textit{Choice} & & \mid\ G \vee G \\
\textit{Sequencing} & & \mid\ G \,;\, G \\
\textit{Guard} & & \mid\ \ominus(\Psi) \\
\textit{Assumption} & & \mid\ \oplus(\Psi) \\
\textit{Inaction} & & \mid\ \mathtt{emp}
\end{array}
$$

$$(\textit{Parties})\ \mathtt{P}, \mathtt{S}, \mathtt{R} \in \mathcal{R}\mathtt{ole} \quad (\textit{Channels})\ \mathtt{c} \in \mathcal{C}\mathtt{han} \quad (\textit{Messages})\ \mathtt{v} \cdot \Delta \quad (\textit{Labels})\ i \in \mathtt{Nat}$$

Given a global protocol G,

1. collect all the event orderings as guards and assumptions, and
2. refine G to account for the guards and assumptions.

1. Collecting Ordering Assumptions

Communicates-before between the sending and receiving events*:*

$\mathtt{S} \xrightarrow{i} \mathtt{R} : \mathtt{c}\langle \mathtt{v} \cdot \Delta \rangle$

Local events

Global orderings

$$\oplus(\mathtt{S} \xrightarrow{i} \mathtt{R} : \mathtt{c}\langle \mathtt{v} \cdot \Delta \rangle) \Rightarrow \boxed{\oplus(\mathtt{S}^{(i)})} \wedge \boxed{\oplus(\mathtt{R}^{(i)})} \wedge \oplus(\mathtt{S}^{(i)} \prec_{\mathtt{CB}} \mathtt{R}^{(i)})$$

Happens-before between events on the same party P (program order)*:*

$\mathtt{P} \xrightarrow{i_1} \mathtt{R}_1 : \mathtt{c}_1\langle \mathtt{v} \cdot \Delta_1 \rangle; \;\ldots\; ; \mathtt{P} \xrightarrow{i_2} \mathtt{R}_2 : \mathtt{c}_2\langle \mathtt{v} \cdot \Delta_2 \rangle$

$\mathtt{P} \xrightarrow{i_1} \mathtt{R}_1 : \mathtt{c}_1\langle \mathtt{v} \cdot \Delta_1 \rangle; \;\ldots\; ; \mathtt{S}_2 \xrightarrow{i_2} \mathtt{P} : \mathtt{c}_2\langle \mathtt{v} \cdot \Delta_2 \rangle$

$\mathtt{S}_1 \xrightarrow{i_1} \mathtt{P} : \mathtt{c}_1\langle \mathtt{v} \cdot \Delta_1 \rangle; \;\ldots\; ; \mathtt{P} \xrightarrow{i_2} \mathtt{R}_2 : \mathtt{c}_2\langle \mathtt{v} \cdot \Delta_2 \rangle$

$\mathtt{S}_1 \xrightarrow{i_1} \mathtt{P} : \mathtt{c}_1\langle \mathtt{v} \cdot \Delta_1 \rangle; \;\ldots\; ; \mathtt{S}_2 \xrightarrow{i_2} \mathtt{P} : \mathtt{c}_2\langle \mathtt{v} \cdot \Delta_2 \rangle$

$$\oplus(\mathtt{P}^{(i_1)} \prec_{\mathtt{HB}} \mathtt{P}^{(i_2)})$$

1. Collecting Ordering Guards

Theorem: Race-free Protocol

A protocol G is race-free if and only if all the adjacent transmissions are race-free:

$$(\forall i_1, i_2 \in G \cdot \mathtt{Adj}(i_1, i_2) \Rightarrow \mathtt{RF}(i_1, i_2)) \Leftrightarrow \mathtt{RF}(G).$$

$$\ldots; \mathtt{S}_1 \xrightarrow{\mathtt{i}_1} \mathtt{R}_1 : \mathtt{c}\langle \Delta_1 \rangle; \;\ldots\; ; \mathtt{S}_2 \xrightarrow{\mathtt{i}_2} \mathtt{R}_2 : \mathtt{c}\langle \Delta_2 \rangle; \ldots$$

Proof-obligation to check race-freedom:

$$\ominus(i_1 \prec_{\mathtt{HB}} i_2)$$

2. Protocol Refinement

$$(\mathtt{S} \xrightarrow{2} \mathtt{A} : \mathtt{a}\langle \mathtt{v} \cdot \mathtt{v} > 0 \rangle * \mathtt{S} \xrightarrow{3} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} > 0 \rangle) ; \mathtt{A} \xrightarrow{4} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} \geq 0 \rangle$$

Refinement
(automatically)

$$\begin{aligned}
&(\mathtt{S} \xrightarrow{2} \mathtt{A} : \mathtt{a}\langle \mathtt{v} \cdot \mathtt{v} > 0 \rangle ; \oplus(\mathtt{S}^{(2)}) ; \oplus(\mathtt{A}^{(2)}) ; \oplus(\mathtt{S}^{(2)} \prec_{\mathtt{CB}} \mathtt{A}^{(2)}) * \\
&\ \mathtt{S} \xrightarrow{3} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} > 0 \rangle ; \oplus(\mathtt{S}^{(3)}) ; \oplus(\mathtt{B}^{(3)}) ; \oplus(\mathtt{S}^{(3)} \prec_{\mathtt{CB}} \mathtt{B}^{(3)})) ; \\
&\mathtt{A} \xrightarrow{4} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} \geq 0 \rangle ; \oplus(\mathtt{A}^{(4)}) ; \oplus(\mathtt{B}^{(4)}) ; \oplus(\mathtt{A}^{(4)} \prec_{\mathtt{CB}} \mathtt{B}^{(4)}) ; \\
&\qquad \oplus(\mathtt{A}^{(2)} \prec_{\mathtt{HB}} \mathtt{A}^{(4)}) ; \oplus(\mathtt{B}^{(3)} \prec_{\mathtt{HB}} \mathtt{B}^{(4)}) ; \\
&\qquad \ominus(3 \prec_{\mathtt{HB}} 4)
\end{aligned}$$

Take – away 4: RELAXED PROTOCOLS

2D. Modular Protocols

Modular Protocols

$$G_{\mathtt{ABS}} \triangleq \mathtt{A}\xrightarrow{1}\mathtt{S}:\mathtt{s}\langle\mathtt{String}\rangle\,;\ (\mathtt{S}\xrightarrow{2}\mathtt{A}:\mathtt{a}\langle\mathtt{v}\cdot\mathtt{v}>0\rangle * \mathtt{S}\xrightarrow{3}\mathtt{B}:\mathtt{b}\langle\mathtt{v}\cdot\mathtt{v}>0\rangle)\,;\ \mathtt{A}\xrightarrow{4}\mathtt{B}:\mathtt{b}\langle\mathtt{v}\cdot\mathtt{v}\geq 0\rangle\,;\ (\mathtt{B}\xrightarrow{5}\mathtt{S}:\mathtt{s}\langle\mathtt{ok}\rangle\,;\ \mathtt{B}\xrightarrow{6}\mathtt{S}:\mathtt{s}\langle\mathtt{v}\cdot\mathtt{Addr(v)}\rangle\,;\ \mathtt{S}\xrightarrow{7}\mathtt{B}:\mathtt{b}\langle\mathtt{v}\cdot\mathtt{Date(v)}\rangle \vee \mathtt{B}\xrightarrow{8}\mathtt{S}:\mathtt{s}\langle\mathtt{quit}\rangle).$$

Refinement

(automatically)

$$\overline{G}_{\mathtt{ABS}} \triangleq \ldots$$

1. Make protocols instantiable by treating them as abstract predicates with **parameters**.

$$G_{\mathtt{ABS}} \triangleq \ldots \longrightarrow G_{\mathtt{ABS}}(\mathtt{A},\mathtt{B},\mathtt{S},\mathtt{a},\mathtt{b},\mathtt{s}) \triangleq \ldots$$

2. Attach a labelling system which contains **instantiable labels** and maintains uniqueness of transmissions.

$$G_{\mathtt{ABS}}(\mathtt{A},\mathtt{B},\mathtt{S},\mathtt{a},\mathtt{b},\mathtt{s}) \triangleq \ldots \longrightarrow G_{\mathtt{ABS}}(\mathtt{A},\mathtt{B},\mathtt{S},\mathtt{a},\mathtt{b},\mathtt{s},i) \triangleq \mathtt{A}\xrightarrow{i\#1}\mathtt{S}:\mathtt{s}\langle\mathtt{String}\rangle\,;\ (\mathtt{S}\xrightarrow{i\#2}\mathtt{A}:\mathtt{a}\langle\mathtt{v}\cdot\mathtt{v}>0\rangle * \mathtt{S}\xrightarrow{i\#3}\mathtt{B}:\mathtt{b}\langle\mathtt{v}\cdot\mathtt{v}>0\rangle)\,;\ \ldots$$

3. Create event ordering summaries for each predicate (HB relations between the first and last encounter of each communicating party).

4. Synthesize the necessary conditions for a safe synchronization with the environment.

Outline of the talk

1. Related Work

2. Session Logic

   A. Specification Language

   B. Identify Race Conditions

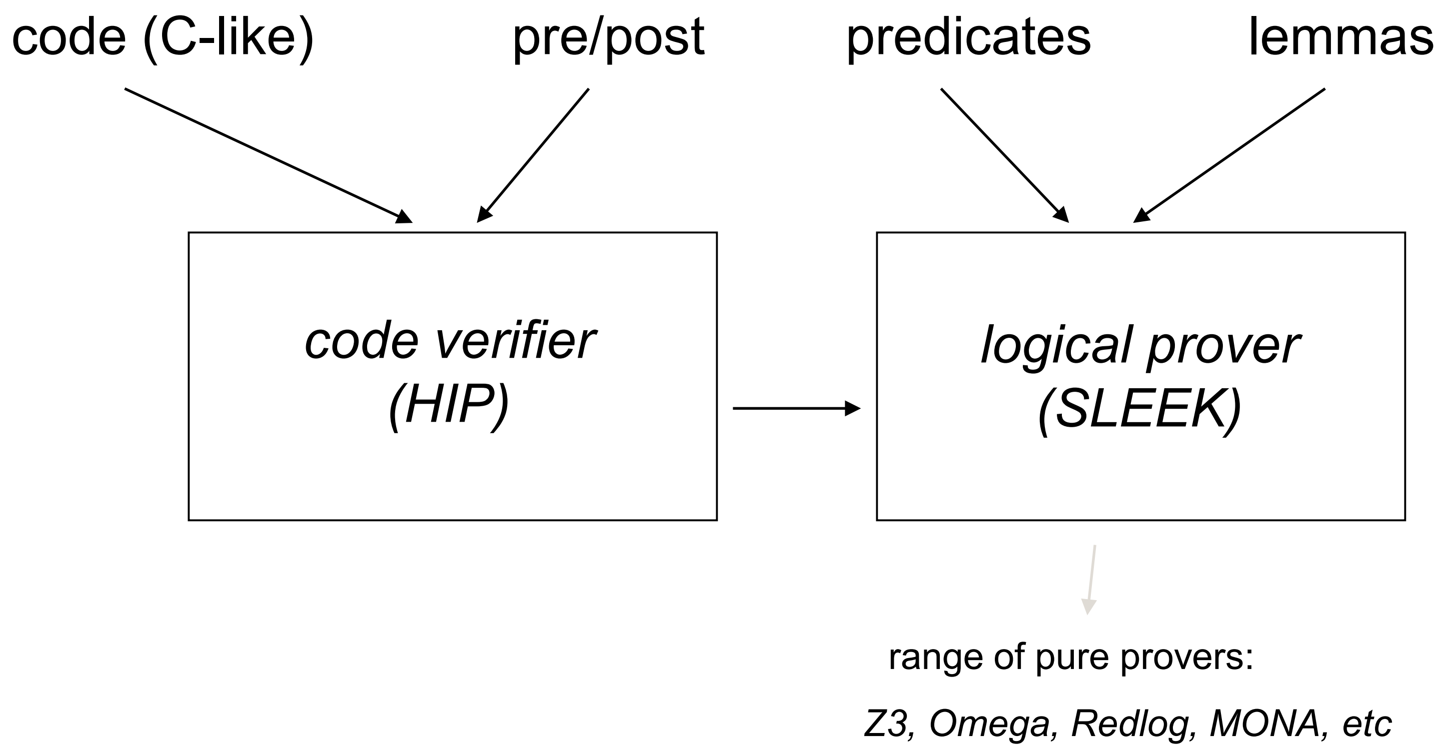
   C. Relaxed Protocols

   D. Modular Protocols

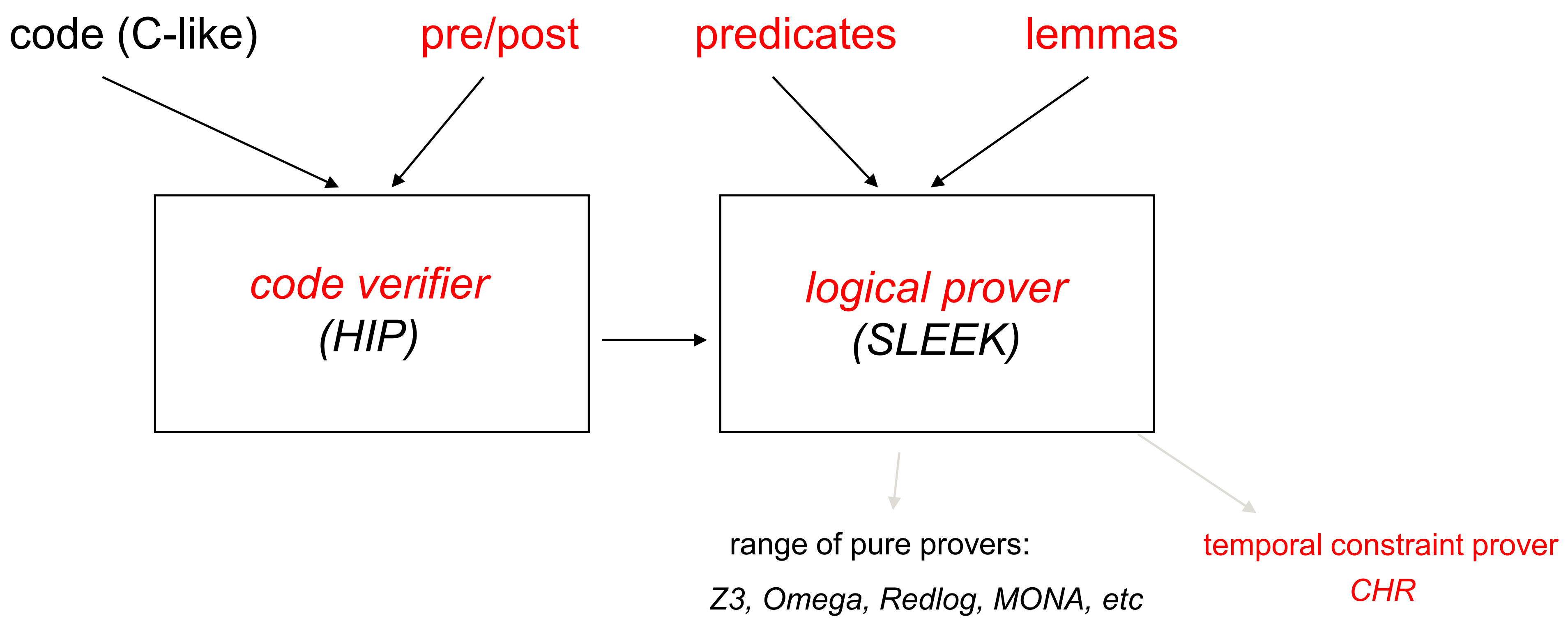
3. Communication Verification

4. Conclusion and Future Work

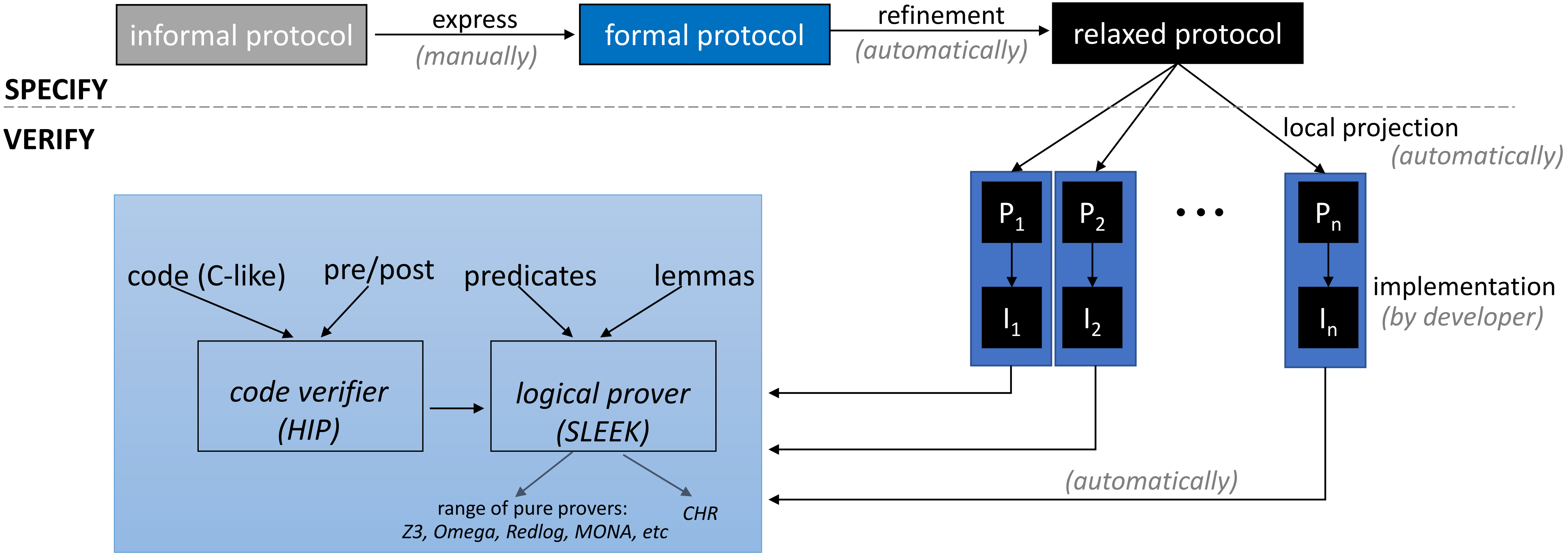
Verification Framework



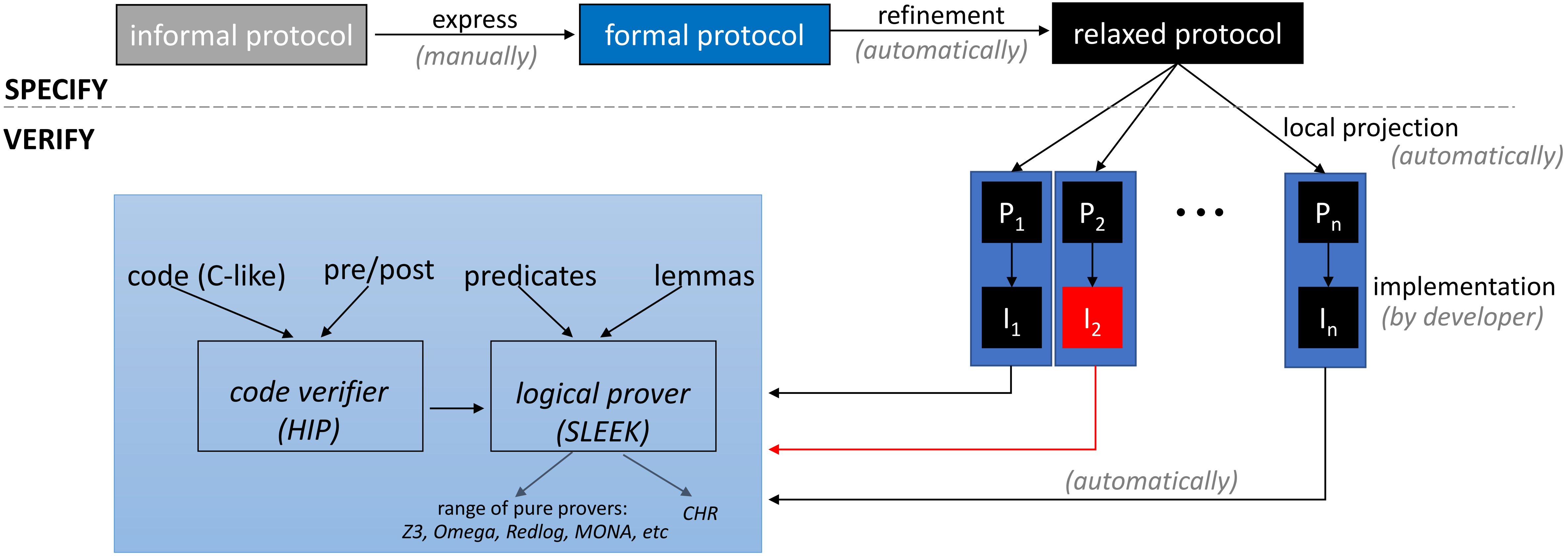
Verification Framework



Framework Overview



Framework Overview



Local Projection

$$
\begin{array}{llll}
\textit{Global protocol} & G & ::= & \\
\textit{Single transmission} & & & \mathtt{S} \xrightarrow{i} \mathtt{R} : \mathtt{c}\langle \mathtt{v} \cdot \Delta \rangle \\
\textit{Concurrency} & & & \mid G * G \\
\textit{Choice} & & & \mid G \vee G \\
\textit{Sequencing} & & & \mid G \,;\, G \\
\textit{Guard} & & & \mid \ominus(\Psi) \\
\textit{Assumption} & & & \mid \oplus(\Psi) \\
\textit{Inaction} & & & \mid \mathtt{emp}
\end{array}
$$

per-party projection

(automatically)

$$
\begin{array}{ll}
\Upsilon ::= & \\
& \mathtt{c!v} \cdot \Delta \mid \mathtt{c?v} \cdot \Delta \\
& \mid \Upsilon * \Upsilon \\
& \mid \Upsilon \vee \Upsilon \\
& \mid \Upsilon ; \Upsilon \\
& \mid \ominus(\Psi) \\
& \mid \oplus(\Psi) \\
& \mid \mathtt{emp}
\end{array}
$$

per channel projection

(automatically)

$$
\begin{array}{ll}
\mathtt{L} ::= & \\
& \mathtt{!v} \cdot \Delta \mid \mathtt{?v} \cdot \Delta \\
& \mid \mathtt{L} \vee \mathtt{L} \\
& \mid \mathtt{L};\mathtt{L} \\
& \mid \ominus(\Psi) \\
& \mid \oplus(\Psi) \\
& \mid \mathtt{emp}
\end{array}
$$

Local Projection

			per party projection (automatically)		per channel projection (automatically)	
Global protocol	G ::=			Υ ::=		L ::=
Single transmission		$S \xrightarrow{i} R : c\langle v \cdot \Delta\rangle$		$c!v \cdot \Delta \mid c?v \cdot \Delta$		$!v \cdot \Delta \mid ?v \cdot \Delta$
Concurrency		$\mid G * G$		$\mid \Upsilon * \Upsilon$		
Choice		$\mid G \vee G$		$\mid \Upsilon \vee \Upsilon$		$\mid L \vee L$
Sequencing		$\mid G \,;\, G$		$\mid \Upsilon ; \Upsilon$		$\mid L;L$
Guard		$\mid \ominus(\Psi)$	→	$\mid \ominus(\Psi)$		$\mid \ominus(\Psi)$
Assumption		$\mid \oplus(\Psi)$		$\mid \oplus(\Psi)$		$\mid \oplus(\Psi)$
Inaction		$\mid$ emp		$\mid$ emp		$\mid$ emp

$$(\ominus(P_1^{(i_1)} \prec_{HB} P_2^{(i_2)}))\rfloor_P \;:=\; \begin{cases} \ominus(P_1^{(i_1)} \prec_{HB} P_2^{(i_2)}) & \text{if } P{=}P_2 \\ \oplus(P_1^{(i_1)} \prec_{HB} P_2^{(i_2)}) & \text{if } P{\neq}P_2 \end{cases}$$

Take – away 5: COLLABORATIVE PROVING

Race-free protocol:

$$(S \xrightarrow{2} A : a\langle v \cdot v > 0\rangle \,;\, \oplus(S^{(2)}) \,;\, \oplus(A^{(2)}) \,;\, \oplus(S^{(2)} \prec_{CB} A^{(2)}) \,*$$
$$S \xrightarrow{3} B : b\langle v \cdot v > 0\rangle \,;\, \oplus(S^{(3)}) \,;\, \oplus(B^{(3)}) \,;\, \oplus(S^{(3)} \prec_{CB} B^{(3)})) \,;$$
$$A \xrightarrow{4} B : b\langle v \cdot v \geq 0\rangle \,;\, \oplus(A^{(4)}) \,;\, \oplus(B^{(4)}) \,;\, \oplus(A^{(4)} \prec_{CB} B^{(4)}) \,;$$
$$\oplus(A^{(2)} \prec_{HB} A^{(4)}) \,;\, \oplus(B^{(3)} \prec_{HB} B^{(4)}) \,;$$
$$\ominus(3 \prec_{HB} 4)$$

$$3 \prec_{HB} 4 \;\equiv\; S^{(3)} \prec_{HB} A^{(4)} \;\wedge\; B^{(3)} \prec_{HB} B^{(4)}.$$

$$(\ominus(3 \prec_{HB} 4))\rfloor_A \;=\; \ominus(S^{(3)} \prec_{HB} A^{(4)}) \;;\; \oplus(B^{(3)} \prec_{HB} B^{(4)}).$$
$$(\ominus(3 \prec_{HB} 4))\rfloor_B \;=\; \oplus(S^{(3)} \prec_{HB} A^{(4)}) \;;\; \ominus(B^{(3)} \prec_{HB} B^{(4)}).$$
$$(\ominus(3 \prec_{HB} 4))\rfloor_S \;=\; \oplus(S^{(3)} \prec_{HB} A^{(4)}) \;;\; \oplus(B^{(3)} \prec_{HB} B^{(4)}).$$

Local Projection

per party projection → (automatically)

per channel projection → (automatically)

Global protocol	G ::=	Υ ::=	L ::=
Single transmission	$\mathtt{S} \xrightarrow{i} \mathtt{R} : \mathtt{c}\langle \mathtt{v} \cdot \Delta \rangle$	$\mathtt{c!v} \cdot \Delta \mid \mathtt{c?v} \cdot \Delta$	$\mathtt{!v} \cdot \Delta \mid \mathtt{?v} \cdot \Delta$
Concurrency	$\mid G * G$	$\mid \Upsilon * \Upsilon$	
Choice	$\mid G \vee G$	$\mid \Upsilon \vee \Upsilon$	$\mid \mathtt{L} \vee \mathtt{L}$
Sequencing	$\mid G \,;\, G$	$\mid \Upsilon ; \Upsilon$	$\mid \mathtt{L};\mathtt{L}$
Guard	$\mid \ominus(\Psi)$	$\mid \ominus(\Psi)$	$\mid \ominus(\Psi)$
Assumption	$\mid \oplus(\Psi)$	$\mid \oplus(\Psi)$	$\mid \oplus(\Psi)$
Inaction	$\mid \mathtt{emp}$	$\mid \mathtt{emp}$	$\mid \mathtt{emp}$

SPECIFY

VERIFY

HO predicate example:

$\mathcal{C}(\mathtt{c}, \mathtt{P}, \mathtt{L})$ - associates a specification $\mathtt{L}$ to a channel $\mathtt{c}$ which is manipulated by party $\mathtt{P}$.

$$\begin{array}{lll} [\mathbf{L}+] & \mathcal{C}(\mathtt{c}, \mathtt{P}, \oplus(\Psi); \mathtt{L}) & \mapsto \ \mathcal{C}(\mathtt{c}, \mathtt{P}, \mathtt{L}) \wedge \Psi. \\ [\mathbf{L}-] & \mathcal{C}(\mathtt{c}, \mathtt{P}, \ominus(\Psi); \mathtt{L}) \wedge \Psi & \mapsto \ \mathcal{C}(\mathtt{c}, \mathtt{P}, \mathtt{L}). \end{array}$$

Local Projection

			per party projection (automatically)			per channel projection (automatically)		
Global protocol	G ::=		$\Longrightarrow$	Υ ::=		$\Longrightarrow$	L ::=	
Single transmission		$\mathtt{S}\xrightarrow{i}\mathtt{R} : \mathtt{c}\langle \mathtt{v}\cdot\Delta\rangle$			$\mathtt{c!v}\cdot\Delta \mid \mathtt{c?v}\cdot\Delta$			$\mathtt{!v}\cdot\Delta \mid \mathtt{?v}\cdot\Delta$
Concurrency		$\mid G * G$			$\mid \Upsilon * \Upsilon$			
Choice		$\mid G \vee G$			$\mid \Upsilon \vee \Upsilon$			$\mid \mathtt{L} \vee \mathtt{L}$
Sequencing		$\mid G\ ;\ G$			$\mid \Upsilon;\Upsilon$			$\mid \mathtt{L};\mathtt{L}$
Guard		$\mid \ominus(\Psi)$			$\mid \ominus(\Psi)$			$\mid \ominus(\Psi)$
Assumption		$\mid \oplus(\Psi)$			$\mid \oplus(\Psi)$			$\mid \oplus(\Psi)$
Inaction		$\mid \mathtt{emp}$			$\mid \mathtt{emp}$			$\mid \mathtt{emp}$

SPECIFY

VERIFY

$$\overline{G}_{\mathtt{ABS}} \triangleq \mathtt{A}\xrightarrow{1}\mathtt{S} : \mathtt{s}\langle\mathtt{String}\rangle;\ \overline{G}$$

$$\xrightarrow{\ \mathsf{A}\ } \mathtt{s!v}\cdot\mathtt{v{:}String};\ \Upsilon_{\mathtt{A}} \xrightarrow{\ \mathsf{A,\ s}\ } \mathtt{!v}\cdot\mathtt{v{:}String};\ \mathtt{L}_{\mathtt{A}}$$

$$\xrightarrow{\ \mathsf{S}\ } \mathtt{s?v}\cdot\mathtt{v{:}String};\ \Upsilon_{\mathtt{S}} \xrightarrow{\ \mathsf{S,\ s}\ } \mathtt{?v}\cdot\mathtt{v{:}String};\ \mathtt{L}_{\mathtt{S}}$$

Communication Primitives

$$\frac{}{\vdash \{\texttt{true}\}\ \texttt{open()}\ \texttt{with}\ (\texttt{c}, \texttt{P}^*)\ \{\ \texttt{opened}(\texttt{c}, \texttt{P}^*, \texttt{res})\}}\ [\textbf{OPEN}]$$

$$\frac{}{\vdash \{\ \texttt{empty}(\tilde{\texttt{c}})\}\ \texttt{close}(\tilde{\texttt{c}})\ \{\ \texttt{true}\ \}}\ [\textbf{CLOSE}]$$

$$\frac{\texttt{inv} \triangleq \texttt{Peer}(\texttt{P}) \wedge \texttt{opened}(\texttt{c}, \texttt{P}^*, \tilde{\texttt{c}}) \wedge \texttt{P} {\in} \texttt{P}^*}{\vdash \{\mathcal{C}(\texttt{c}, \texttt{P}, !\texttt{v} \cdot \texttt{V}(\texttt{v}); \texttt{L}) * \texttt{V}(\texttt{x}) * \texttt{inv}\}\ \texttt{send}(\tilde{\texttt{c}}, \texttt{x})\ \{\mathcal{C}(\texttt{c}, \texttt{P}, \texttt{L}) * \texttt{inv}\}}\ [\textbf{SEND}]$$

$$\frac{\texttt{inv} \triangleq \texttt{Peer}(\texttt{P}) \wedge \texttt{opened}(\texttt{c}, \texttt{P}^*, \tilde{\texttt{c}}) \wedge \texttt{P} {\in} \texttt{P}^*}{\vdash \{\mathcal{C}(\texttt{c}, \texttt{P}, ?\texttt{v} \cdot \texttt{V}(\texttt{v}); \texttt{L}) * \texttt{inv}\}\ \texttt{recv}(\tilde{\texttt{c}})\ \{\mathcal{C}(\texttt{c}, \texttt{P}, \texttt{L}) * \texttt{V}(\texttt{res}) * \texttt{inv}\}}\ [\textbf{RECV}]$$

Collaborative Client – Server (revisited)

Buyer A

```
int price,share;
String book;
…
send(s, book);
price = receive(a);
share = foo(price);
send(b, share);
…
```

Buyer B

```
int price,clb;

…
price = receive(b);
clb = receive(b);
if(cond){
  send(s, ok);
  send(s, addr);
  … = receive(s);
}else {
 send(s, quit);
}
…
```

Seller

```
int id, val;

…
id = receive(s);
val = goo(id);
send(a,val);
send(b,val);
ans = receive(s);
if (s==ok){
  … = receive(s);
  send(b,…);
}
…
```

Collaborative Client – Server (revisited)

Buyer A

```
int price,share;
String book;
…
```

$//\Phi * \mathcal{C}(\mathtt{s}, \mathtt{A}, !\mathtt{v} \cdot \mathtt{v}{:}\mathtt{String}; \mathtt{L}) \wedge \mathtt{book}{:}\mathtt{String}$

```
send(s, book);
```

$//\Phi * \mathcal{C}(\mathtt{s}, \mathtt{A}, \mathtt{L}) \wedge \mathtt{book}{:}\mathtt{String}$

```
price = receive(a);
share = foo(price);
send(b, share);
…
```

Seller

```
int id, val;

…
```

$//\Phi * \mathcal{C}(\mathtt{s}, \mathtt{S}, ?\mathtt{v} \cdot \mathtt{v}{:}\mathtt{String}; \mathtt{L}) \wedge \mathtt{id}{:}\mathtt{int}$

```
id = receive(s);

val = goo(id);
send(a,val);
send(b,val);
ans = receive(s);
if (s==ok){
  … = receive(s);
  send(b,…);
}
…
```

Race Handling (revisited)

Buyer A	Buyer B	Seller
…	…	…
(1) $A^{(4)}$: `send(b, share);`	$B^{(3)}$: `price = receive(b);` $B^{(4)}$: `clb = receive(b);`	$S^{(3)}$: `send(b,val);`
$//\mathcal{C}(b, A, \ominus(S^{(3)} \prec_{HB} A^{(4)}); L_A)$	$//\mathcal{C}(b, B, \ominus(B^{(3)} \prec_{HB} B^{(4)}); L_B)$ $//\mathcal{C}(b, B, L_B)$ ✓	

$$\oplus(S^{(2)} \prec_{CB} A^{(2)})$$
$$\oplus(S^{(3)} \prec_{CB} B^{(3)})$$
$$\oplus(A^{(4)} \prec_{CB} B^{(4)})$$
$$\oplus(A^{(2)} \prec_{HB} A^{(4)}) \quad \oplus(B^{(3)} \prec_{HB} B^{(4)})$$

Global Store

$$(\ominus(3 \prec_{HB} 4))\lfloor_A \;=\; \ominus(S^{(3)} \prec_{HB} A^{(4)}) \;;\; \oplus(B^{(3)} \prec_{HB} B^{(4)}).$$
$$(\ominus(3 \prec_{HB} 4))\lfloor_B \;=\; \oplus(S^{(3)} \prec_{HB} A^{(4)}) \;;\; \ominus(B^{(3)} \prec_{HB} B^{(4)}).$$
$$(\ominus(3 \prec_{HB} 4))\lfloor_S \;=\; \oplus(S^{(3)} \prec_{HB} A^{(4)}) \;;\; \oplus(B^{(3)} \prec_{HB} B^{(4)}).$$

Race free proof obligation projected onto each party

Race Handling (revisited)

Buyer A

(2)

```
      ...
      wait(cnd);
A(4): send(b, share);
```

$//\mathcal{C}(\mathtt{b}, \mathtt{A}, \ominus(\mathtt{S}^{(3)} \prec_{\mathtt{HB}} \mathtt{A}^{(4)}); \mathtt{L_A})$

$//\mathcal{C}(\mathtt{b}, \mathtt{A}, \mathtt{L_A})$ ✓

Buyer B

```
      ...
B(3): price = receive(b);
      notifyAll(cnd);
B(4): clb = receive(b);
```

$//\mathcal{C}(\mathtt{b}, \mathtt{B}, \ominus(\mathtt{B}^{(3)} \prec_{\mathtt{HB}} \mathtt{B}^{(4)}); \mathtt{L_B})$

$//\mathcal{C}(\mathtt{b}, \mathtt{B}, \mathtt{L_B})$ ✓

Seller

```
      ...
S(3): send(b,val);
```

Implementation

In OCaml, affixed to HIP/SLEEK.

The constraint ordering system is implemented in CHR.

Highly modular:

- The protocol components are encoded as higher order primitive predicates.
- The predicates are manipulated by user-defined lemmas.

$\Rightarrow$ finely “tunable” logic to cope with future extensions.

Test cases : variation of client-server, variations of the collaborative client – server, atm, vending machine, video streaming.

Outline of the talk

1. Related Work
2. Session Logic
   A. Specification Language
   B. Identify Race Conditions
   C. Relaxed Protocols
   D. Modular Protocols
3. Communication Verification
4. **Conclusion and Future Work**

We provide a novel theory and necessary tools to specify and reason about distributed systems!

We have shown how to:

… move from **types systems → logic** (going beyond type safety)

… achieve **composable verification** of safety (type-safe, race-free)

via *local projection* and *collaborative proving.*

… ensure **temporal ordering**, without the explicit concept of time

… support **relaxed** and **modular protocols**:

realistic non-linear protocols → race-free protocols with explicit synchronization

A Language-Based Approach to Formalizing Protocols

Thesis:

Language support makes it possible:

- to **specify** communication protocols, and then
- to **verify** (automatically) that an implementation conforms to the given protocol in a safe way.

Beyond This Talk

More in the dissertation:

- a *dyadic session logic* which emphasizes the benefits of going beyond traditional type check: disjunction to replace internal/external choices, higher order-channels, copy and copyless-message passing, deadlock detection, delegation.
- *multiparty session logic:* safety (wrt conformance, race, deadlock) theorems with soundness proofs, detailed verification examples, nondeterminism, efficient algorithm for collecting ordering assertions, inference algorithm for synchronization with the context, recursion, delegation, verification rules, entailment rules, explicit synchronization primitives.

Future work:

- synthesize the specifications for the explicit synchronization mechanisms.
- investigate the formalization of additional properties: consensus of distributed systems.

Thank you!

BIBLIOGRAPHY

BELL , C. J., APPEL , A. W., and WALKER , D., “Concurrent Separation Logic for Pipelined Parallelization,” in SAS 2010, pp. 151–166, Springer.

CAIRES, L., “Spatial-Behavioral Types for Concurrency and Resource Control in Distributed Systems,” Theoretical Computer Science, vol. 402, no. 2-3, pp. 120–141, 2008

CAIRES , L. and PFENNING, F., “Session Types as Intuitionistic Linear Propositions”, in CONCUR’10.

CAIRES , L. and SECO , J. C., “The Type Discipline of Behavioral Separation,” in POPL 2013.

CAIRES, L. and VIEIRA, H. T., “Conversation Types,” in European Symposium on Programming, pp. 285–300, Springer, 2009.

CAPECCHI, S., GIACHINO, E., and YOSHIDA, N., “Global Escape in Multiparty Sessions,” Mathematical Structures in Computer Science, vol. 26, no. 02, pp. 156–205, 2016

CARBONE, M., HONDA, K., and YOSHIDA, N., “Structured Interactional Exceptions in Session Types,” in International Conference on Concurrency Theory, pp. 402–417, Springer, 2008.

CARBONE, M. and MONTESI, F., “Deadlock-freedom-by-design: Multiparty Asynchronous Global Programming,” SIGPLAN Not., vol. 48, pp. 263–274, Jan. 2013.

COPPO, M., DEZANI-CIANCAGLINI, M., YOSHIDA, N., and PADOVANI, L., “Global Progress for Dynamically Interleaved Multiparty Sessions,” Mathematical Structures in Computer Science, vol. 26, no. 02, pp. 238–302, 2016.

DEMANGEON, R., HONDA, K., “Nested Protocols in Session Types”, 23rd International Conference on Concurrency Theory (CONCUR 2012)

DENIELOU, P.-M. and YOSHIDA, N., “Buffered Communication Analysis in Distributed Multiparty Sessions,” in CONCUR 2010, vol. 6269 of LNCS, Springer, 2010.

DENIÉLOU, P.-M. and YOSHIDA, N., “Dynamic multirole session types,” in Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL ’11, (New York, NY, USA), pp. 435–446, ACM, 2011.

DEZANI-CIANCAGLINI, M., MOSTROUS, D., YOSHIDA, N., and DROSSOPOULOU, S., “Session types for object-oriented languages,” in Proceedings of the 20th European Conference on Object-Oriented Programming, ECOOP’06, (Berlin, Heidelberg), pp. 328–352, Springer-Verlag, 2006.

GAY, S. and HOLE, M., “Subtyping for Session Types in the pi-Calculus,” Acta Informatica, vol. 42, no. 2-3, pp. 191–225, 2005.

GAY, S. J. and VASCONCELOS, V. T., “Linear Type Theory for Asynchronous Session Types,” Journal of Functional Programming, vol. 20, no. 01, pp. 19–50, 2010.

GIUNTI, M. and VASCONCELOS, V. T., “A Linear Account of Session Types in the Pi Calculus”, in CONCUR 2010

HOARE, T. and O’HEARN, P., “Separation Logic Semantics for Communicating Processes,” Electronic Notes in Theoretical Computer Science, vol. 212, pp. 3–25, 2008.

HONDA, K., “Composing Processes,” in Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of programming languages, pp. 344–357, ACM, 1996.

HONDA , K., VASCONCELOS , V. T., and KUBO , M., “Language primitives and type discipline for structured communication-based programming,” in ESOP ’98.

HONDA , K., YOSHIDA , N., and CARBONE , M., “Multiparty Asynchronous Session Types,” POPL 2008.

HONDA, K., YOSHIDA, N., and CARBONE, M., “Multiparty Asynchronous Session Types,” Journal of the ACM, vol. 63, pp. 1–67, 2016.

HU, R., YOSHIDA, N., “Hybrid Session Verification Through Endpoint API Generation”, Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering - Volume 9633 , 2016

IGARASHI , A. and KOBAYASHI , N., “A Generic Type System for the Pi-Calculus,” Theoretical Computer Science, vol. 311, no. 1, pp. 121 – 163, 2004.

KOBAYASHI, N., “Type Systems for Concurrent Processes: From Deadlock-freedom to Livelock-freedom, Time-boundedness,” in IFIP International Conference on Theoretical Computer Science, pp. 365–389, Springer, 2000.

KOBAYASHI, N., “A Type System for Lock-free Processes,” Information and Computation, vol. 177, no. 2, pp. 122–159, 2002.

KOBAYASHI, N., “Type Systems for Concurrent Programs,” in Formal Methods at the Crossroads. From Panacea to Foundational Support, pp. 439–453, Springer, 2003.

KOBAYASHI, N., “Type-based Information Flow Analysis for the -calculus,” Acta Informatica, vol. 42, no. 4-5, pp. 291–347, 2005.

KOBAYASHI, N., “A New Type System for Deadlock-free Processes,” in CONCUR 2006–Concurrency Theory, pp. 233–247, Springer Berlin Heidelberg, 2006.

KOBAYASHI, N. and LANEVE, C., “Deadlock Analysis of Unbounded Process Networks,” Information and Computation, vol. 252, pp. 48–70, 2017.

KOUZAPAS, D., YOSHIDA, N., HU, R., and HONDA, K., “On Asynchronous Eventful Session Semantics,” Mathematical Structures in Computer Science, vol. 26, no. 02, pp. 303–364, 2016.

LANGE, J., YOSHIDA, N., “On the Undecidability of Asynchronous Session Subtyping”, ”, Proceedings of the 19th International Conference on Fundamental Approaches to Software Engineering - Volume 9633 , 2016

LEINO , K. R. M., MÜLLER , P., and SMANS , J., “Deadlock-Free Channels and Locks,” in ESOP 2010, pp. 407–426, Springer.

LEINO , K. R. M. and MÜLLER , P., “A Basis for Verifying Multi-Threaded Programs,” in ESOP 2009 pp. 378–393, Springer.

LINDLEY, S. and MORRIS, J. G., “A Semantics for Propositions as Sessions,” in European Symposium on Programming Languages and Systems, pp. 560–584, Springer, 2015.

LINDLEY, S. and MORRIS, J. G., “Embedding Session Types in Haskell,” in Proceedings of the 9th International Symposium on Haskell, pp. 133–145, ACM, 2016.

NEUBAUER, M. and THIEMANN, P., “An Implementation of Session Types,” in International Symposium on Practical Aspects of Declarative Languages, pp. 56–70, Springer, 2004.

NG, N. and YOSHIDA, N., “Pabble: Parameterised Scribble for Parallel Programming,” in Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on, pp. 707–714, IEEE, 2014.

ORCHARD, D. and YOSHIDA, N., “Effects as Sessions, Sessions as Effects,” in ACM SIGPLAN Notices, vol. 51, pp. 568–581, ACM, POPL 2016.

O’HEARN, P. W., “Resources, concurrency, and local reasoning”, Th. Comp. Sci, 375, 2007

PADOVANI, L., “Deadlock and Lock Freedom in the Linear -calculus,” in Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), p. 72, ACM, 2014.

TURON, A. and WAND, M., “A Resource Analysis of the -calculus,” Electronic Notes in Theoretical Computer Science, vol. 276, pp. 313–334, 2011.

VILLARD , J., L OZES , É., and C ALCAGNO , C., “Proving copyless message passing,” in APLAS 2009 , pp. 194–209, Springer.

WADLER, P., “Propositions as Sessions,” ACM SIGPLAN Notices, vol. 47, no. 9, pp. 273–286, 2012.

YOSHIDA, N., HU, R., NEYKOVA, R., NG, N., “The Scribble Protocol Language”, TGC 2013: 8th International Symposium on Trustworthy Global Computing - Volume 8358

Explicit Synchronization

$$\frac{\underline{\textbf{CREATE}}\quad V = \bigwedge_{j \in \{2..n\}} \oplus(E_j \Rightarrow E_1 \prec_{HB} E_j)}{\{emp\}\ w = create()\ with\ E_1, \overline{E_2..E_n}\ \{NOTIFY(w, \ominus(E_1)) * WAIT(w, V)\}}$$

$$\underline{\textbf{NOTIFY-ALL}}$$

$$\{NOTIFY(w, \ominus(E_1)) \wedge E_1\}\ notifyAll(w)\ \{NOTIFY(w, emp)\}$$

$$\frac{\underline{\textbf{WAIT}}\quad V^{rel} = \oplus(E_2 \Rightarrow E_1 \prec_{HB} E_2)}{\{WAIT(w, V^{rel}) \wedge \neg(E_2)\}\ \mathbf{wait(w)}\ \{WAIT(w, emp) * V^{rel}\}}$$

(Wait lemma)	$\oplus(E_2 \Rightarrow E_1 \prec_{HB} E_2) \wedge E_2 \Rightarrow E_1 \prec_{HB} E_2$
(Distribute-waits lemma)	$WAIT(w, \bigwedge_{j \in \{2..n\}} \Psi_j) \Rightarrow \bigwedge_{j \in \{2..n\}} WAIT(w, \Psi_j)$
(Deadlock check)	$NOTIFY(w, \ominus(E_1)) * WAIT(w, emp) \Rightarrow false$

Explicit Synchronization

$$\frac{[\textbf{CREATE}] \quad \mathtt{V} = \bigwedge_{\mathtt{j}\in\{2..n\}} \oplus(\mathtt{E_j} \Rightarrow \mathtt{E_1} \prec_{\mathtt{HB}} \mathtt{E_j})}{\{\mathtt{emp}\}\ \mathtt{w = create()\ with\ E_1}, \overline{\mathtt{E_2..E_n}}\ \{\mathtt{NOTIFY(w, \ominus(E_1)) * WAIT(w, V)}\}}$$

$$[\textbf{NOTIFY-ALL}]$$

$$\{\mathtt{NOTIFY(w, \ominus(E_1)) \wedge E_1}\}\ \mathtt{notifyAll(w)}\ \{\mathtt{NOTIFY(w, emp)}\}$$

$$\frac{[\textbf{WAIT}] \quad \mathtt{V^{rel}} = \oplus(\mathtt{E_2} \Rightarrow \mathtt{E_1} \prec_{\mathtt{HB}} \mathtt{E_2})}{\{\mathtt{WAIT(w, V^{rel}) \wedge \neg(E_2)}\}\ \mathbf{wait(w)}\ \{\mathtt{WAIT(w, emp) * V^{rel}}\}}$$

(Wait lemma)	$\oplus(\mathtt{E_2} \Rightarrow \mathtt{E_1} \prec_{\mathtt{HB}} \mathtt{E_2}) \wedge \mathtt{E_2} \Rightarrow \mathtt{E_1} \prec_{\mathtt{HB}} \mathtt{E_2}$
(Distribute-waits lemma)	$\mathtt{WAIT}(\mathtt{w}, \bigwedge_{\mathtt{j}\in\{2..n\}} \Psi_{\mathtt{j}}) \Rightarrow \bigwedge_{\mathtt{j}\in\{2..n\}} \mathtt{WAIT}(\mathtt{w}, \Psi_{\mathtt{j}})$
(Deadlock check)	$\mathtt{NOTIFY(w, \ominus(E_1)) * WAIT(w, emp)} \Rightarrow \mathtt{false}$

Take – away 5: EXPLICIT SYNCHRONIZATION

Communication Primitives

$$\text{[OPEN]} \quad \vdash \{\ \mathtt{init}(c)\ \}\ \mathtt{open()\ with}\ (c, P^*)\ \{\ \mathtt{opened}(c, P^*, \mathtt{res})\}$$

$$\text{[CLOSE]} \quad \vdash \{\ \mathtt{empty}(\tilde{c})\}\ \mathtt{close}(\tilde{c})\ \{\ \mathtt{emp}\ \}$$

$$\text{[SEND]} \quad \frac{\mathtt{inv} \triangleq \mathtt{Peer}(P) \wedge \mathtt{opened}(c, P^*, \tilde{c}) \wedge P \in P^*}{\vdash \{\mathcal{C}(c, P, !v \cdot V(v); L) * V(x) * \mathtt{inv}\}\ \mathtt{send}(\tilde{c}, x)\ \{\mathcal{C}(c, P, L) * \mathtt{inv}\}}$$

$$\text{[RECV]} \quad \frac{\mathtt{inv} \triangleq \mathtt{Peer}(P) \wedge \mathtt{opened}(c, P^*, \tilde{c}) \wedge P \in P^*}{\vdash \{\mathcal{C}(c, P, ?v \cdot V(v); L) * \mathtt{inv}\}\ \mathtt{recv}(\tilde{c})\ \{\mathcal{C}(c, P, L) * V(\mathtt{res}) * \mathtt{inv}\}}$$

Communication Primitives

$$
\begin{array}{lll}
G(\{\mathtt{P_1..P_n}\}, \mathtt{c}^*) & \mapsto & \mathtt{Party}(\mathtt{P_1}, \mathtt{c}^*, (G)\lfloor_{\mathtt{P_1}}) * ... * \mathtt{Party}(\mathtt{P_n}, \mathtt{c}^*, (G)\lfloor_{\mathtt{P_n}}) \ * \ \mathtt{initall}(\mathtt{c}^*). \\
\mathtt{Party}(\mathtt{P}, \{\mathtt{c_1..c_m}\}, (G)\lfloor_{\mathtt{P}}) & \mapsto & \mathcal{C}(\mathtt{c_1}, \mathtt{P}, (G)\lfloor_{\mathtt{P,c_1}}) * ... * \mathcal{C}(\mathtt{c_m}, \mathtt{P}, (G)\lfloor_{\mathtt{P,c_m}}) * \ \mathtt{Bind}(\mathtt{P}, \{\mathtt{c_1..c_m}\}). \\
\mathtt{initall}(\{\mathtt{c_1..c_m}\}) & \mapsto & \mathtt{init}(\mathtt{c_1}) * ... * \mathtt{init}(\mathtt{c_m}).
\end{array}
$$

(a) Splitting lemmas

$$
\begin{array}{ll}
[\mathbf{EMP-C}] & \mathcal{C}(\mathtt{c}, \mathtt{P_1}, \mathtt{emp}) * ... * \mathcal{C}(\mathtt{c}, \mathtt{P_n}, \mathtt{emp}) \ \wedge \ \mathtt{opened}(\mathtt{c}, \{\mathtt{P_1..P_n}\}, \tilde{\mathtt{c}}) \ \mapsto \ \mathtt{empty}(\tilde{\mathtt{c}}). \\
[\mathbf{EMP-P}] & \mathcal{C}(\mathtt{c_1}, \mathtt{P}, \mathtt{emp}) * ... * \mathcal{C}(\mathtt{c_m}, \mathtt{P}, \mathtt{emp}) \ * \ \mathtt{Bind}(\mathtt{P}, \{\mathtt{c_1..c_m}\}) \ \mapsto \ \mathtt{Party}(\mathtt{P}, \mathtt{c}^*, \mathtt{emp}).
\end{array}
$$

(b) Joining lemmas

$$
\begin{array}{llll}
[\mathbf{L+}] & \mathcal{C}(\mathtt{c}, \mathtt{P}, \oplus(\Psi); \mathtt{L}) & \mapsto & \mathcal{C}(\mathtt{c}, \mathtt{P}, \mathtt{L}) \wedge \Psi. \\
[\mathbf{L-}] & \mathcal{C}(\mathtt{c}, \mathtt{P}, \ominus(\Psi); \mathtt{L}) \wedge \Psi & \mapsto & \mathcal{C}(\mathtt{c}, \mathtt{P}, \mathtt{L}).
\end{array}
$$

(c) Lemmas to handle orders

Figure 1: Lemmas for Specification Manipulation

Race Handling

$$(\mathtt{S}\xrightarrow{2}\mathtt{A} : \mathtt{a}\langle \mathtt{v}\cdot \mathtt{v} > 0\rangle \ast \mathtt{S}\xrightarrow{3}\mathtt{B} : \mathtt{b}\langle \mathtt{v}\cdot \mathtt{v} > 0\rangle)\,;\, \mathtt{A}\xrightarrow{4}\mathtt{B} : \mathtt{b}\langle \mathtt{v}\cdot \mathtt{v} \geq 0\rangle$$

Client A	Client B	Server
…	…	…
(1) `wait(cnd);`	$B^{(3)}$: `price = receive(b);`	$S^{(3)}$: `send(b,val);`
$A^{(4)}$: `send(b, share);`	`notifyAll(cnd);`	
	$B^{(4)}$: `clb = receive(b);`	

$$\begin{aligned}
(\ominus(3 \prec_{\mathrm{HB}} 4))\!\restriction_{\mathtt{A}} \;&=\; \ominus(\mathtt{S}^{(3)} \prec_{\mathrm{HB}} \mathtt{A}^{(4)})\;;\;\oplus(\mathtt{B}^{(3)} \prec_{\mathrm{HB}} \mathtt{B}^{(4)}).\\
(\ominus(3 \prec_{\mathrm{HB}} 4))\!\restriction_{\mathtt{B}} \;&=\; \oplus(\mathtt{S}^{(3)} \prec_{\mathrm{HB}} \mathtt{A}^{(4)})\;;\;\ominus(\mathtt{B}^{(3)} \prec_{\mathrm{HB}} \mathtt{B}^{(4)}).\\
(\ominus(3 \prec_{\mathrm{HB}} 4))\!\restriction_{\mathtt{S}} \;&=\; \oplus(\mathtt{S}^{(3)} \prec_{\mathrm{HB}} \mathtt{A}^{(4)})\;;\;\oplus(\mathtt{B}^{(3)} \prec_{\mathrm{HB}} \mathtt{B}^{(4)}).
\end{aligned}$$

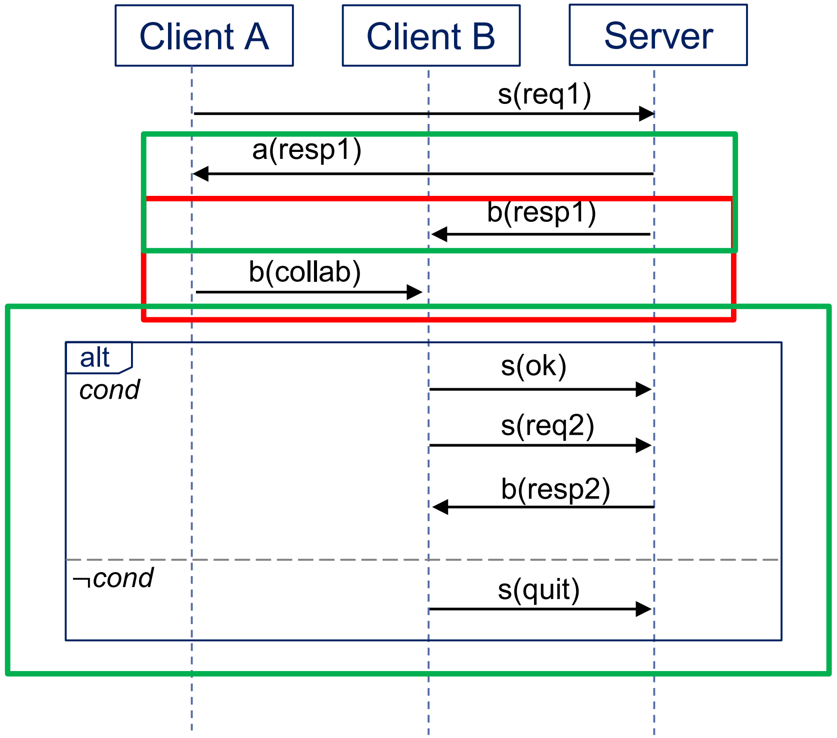
Race free proof obligation projected onto each party

$\{\neg(\mathtt{A}^{(4)})\}$ `wait(cnd);` $\{\mathtt{A}^{(4)} \Rightarrow \mathtt{B}^{(3)} \prec_{\mathrm{HB}} \mathtt{A}^{(4)}\}$

$\{\mathtt{B}^{(3)}\}$ `notifyAll(cnd);` $\{\mathtt{true}\}$

$$\mathtt{S}^{(3)} \prec_{\mathrm{CB}} \mathtt{B}^{(3)} \wedge \mathtt{B}^{(3)} \prec_{\mathrm{HB}} \mathtt{A}^{(4)} \overset{[\text{CB-HB}]}{\Longrightarrow} \mathtt{S}^{(3)} \prec_{\mathrm{HB}} \mathtt{A}^{(4)}$$

Collaborative Client – Server (revisited)



$$
\begin{aligned}
G_{\mathtt{ABS}} \triangleq\ & \mathtt{A} \xrightarrow{1} \mathtt{S} : \mathtt{s}\langle \mathtt{v} \cdot \mathtt{v} : \mathtt{String} \rangle \,; \\
& (\mathtt{S} \xrightarrow{2} \mathtt{A} : \mathtt{a}\langle \mathtt{v} \cdot \mathtt{v} > 0 \rangle \ast \mathtt{S} \xrightarrow{3} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} > 0 \rangle) \,; \mathtt{A} \xrightarrow{4} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{v} \geq 0 \rangle \,; \\
& (\mathtt{B} \xrightarrow{5} \mathtt{S} : \mathtt{s}\langle \mathtt{ok} \rangle \,; \mathtt{B} \xrightarrow{6} \mathtt{S} : \mathtt{s}\langle \mathtt{v} \cdot \mathtt{Addr}(\mathtt{v}) \rangle \,; \mathtt{S} \xrightarrow{7} \mathtt{B} : \mathtt{b}\langle \mathtt{v} \cdot \mathtt{Date}(\mathtt{v}) \rangle \\
& \vee \mathtt{B} \xrightarrow{8} \mathtt{S} : \mathtt{s}\langle \mathtt{quit} \rangle).
\end{aligned}
$$

<u>Different from session types:</u>

1. Messages are described by *logical formulae*.
2. *Concurrent/arbitrary-ordered* transmissions.
3. Uniform treatment of internal/external choice via *disjunction*.

*Common pitfall in creating smart contracts: the domain of the receiver does not subsume the domain of the sender.

Take – away 1: TYPE SYSTEMS -> LOGIC

* DELMOLINO et al., "Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab", in Financial Cryptography and Data Security, pp.79-94, 2016

Example 3 - Verification

$$G(\mathtt{A}, \mathtt{B}, \mathtt{C}, \mathtt{c}, \mathtt{d}) \triangleq \mathtt{A} \xrightarrow{1} \mathtt{C} : \mathtt{c}\langle \Delta_1 \rangle \ ; \ \mathtt{A} \xrightarrow{2} \mathtt{B} : \mathtt{d}\langle \Delta_2 \rangle \ ; \ \mathtt{B} \xrightarrow{3} \mathtt{C} : \mathtt{c}\langle \Delta_3 \rangle$$

$$\{\mathtt{Common}(G\#\mathtt{All}) * \mathtt{Party}(\mathtt{A}, G\#\mathtt{A}) * \mathtt{Party}(\mathtt{B}, G\#\mathtt{B}) * \mathtt{Party}(\mathtt{C}, G\#\mathtt{C})\}$$
$$(\mathtt{Code}_\mathtt{A} \mid\mid \mathtt{Code}_\mathtt{B} \mid\mid \mathtt{Code}_\mathtt{C})$$
$$\{\mathtt{Party}(\mathtt{A}, \mathtt{emp}) * \mathtt{Party}(\mathtt{B}, \mathtt{emp}) * \mathtt{Party}(\mathtt{C}, \mathtt{emp})\}$$

"Release" lemma:

$$\mathtt{Party}(\mathtt{B}, G\#\mathtt{B}) \Rightarrow \mathcal{C}(\mathtt{c}, \mathtt{B}, G\#\mathtt{B}\#\mathtt{c}) * \mathcal{C}(\mathtt{d}, \mathtt{B}, G\#\mathtt{B}\#\mathtt{d})$$

"Join-emp" lemma:

$$\mathtt{Party}(\mathtt{A}, \mathtt{emp}) \Leftrightarrow \mathcal{C}(\mathtt{c}, \mathtt{A}, \mathtt{emp}) * \mathcal{C}(\mathtt{d}, \mathtt{A}, \mathtt{emp})$$

Example 3 - Verification

$$G(\mathtt{A},\mathtt{B},\mathtt{C},\mathtt{c},\mathtt{d}) \triangleq \mathtt{A}\xrightarrow{1}\mathtt{C} : \mathtt{c}\langle\Delta_1\rangle \ ; \ \mathtt{A}\xrightarrow{2}\mathtt{B} : \mathtt{d}\langle\Delta_2\rangle \ ; \ \mathtt{B}\xrightarrow{3}\mathtt{C} : \mathtt{c}\langle\Delta_3\rangle$$

$G\#\mathtt{All} \triangleq \oplus(\mathtt{A}^{(1)} \prec_{\mathtt{CB}} \mathtt{C}^{(1)}); \oplus(\mathtt{A}^{(1)} \prec_{\mathtt{HB}} \mathtt{A}^{(2)}); \oplus(\mathtt{A}^{(2)} \prec_{\mathtt{CB}} \mathtt{B}^{(2)}); \oplus(\mathtt{B}^{(2)} \prec_{\mathtt{HB}} \mathtt{B}^{(3)}); \oplus(\mathtt{C}^{(1)} \prec_{\mathtt{HB}} \mathtt{C}^{(3)}); \oplus(\mathtt{B}^{(3)} \prec_{\mathtt{CB}} \mathtt{C}^{(3)})$

$G\#\mathtt{B}\#\mathtt{c} \triangleq \ominus(\mathtt{B}^{(2)}); ! \cdot \Delta_3; \oplus(\mathtt{B}^{(3)}); \ominus(\mathtt{A}^{(1)} \prec_{\mathtt{HB}} \mathtt{B}^{(3)}); \oplus(\mathtt{C}^{(1)} \prec_{\mathtt{HB}} \mathtt{C}^{(3)})$

$G\#\mathtt{B}\#\mathtt{d} \triangleq ?\Delta_2 \cdot ; \oplus(\mathtt{B}^{(2)})$

```
x = receive(d);

send(c, … );
```

Example 3 – Verification

$$G(\mathtt{A},\mathtt{B},\mathtt{C},\mathtt{c},\mathtt{d}) \triangleq \mathtt{A}\xrightarrow{1}\mathtt{C} : \mathtt{c}\langle\Delta_1\rangle \ ; \ \mathtt{A}\xrightarrow{2}\mathtt{B} : \mathtt{d}\langle\Delta_2\rangle \ ; \ \mathtt{B}\xrightarrow{3}\mathtt{C} : \mathtt{c}\langle\Delta_3\rangle$$

$G\#\mathtt{All} \triangleq \oplus(\mathtt{A}^{(1)} \prec_{\mathtt{CB}} \mathtt{C}^{(1)}); \oplus(\mathtt{A}^{(1)} \prec_{\mathtt{HB}} \mathtt{A}^{(2)}); \oplus(\mathtt{A}^{(2)} \prec_{\mathtt{CB}} \mathtt{B}^{(2)}); \oplus(\mathtt{B}^{(2)} \prec_{\mathtt{HB}} \mathtt{B}^{(3)}); \oplus(\mathtt{C}^{(1)} \prec_{\mathtt{HB}} \mathtt{C}^{(3)}); \oplus(\mathtt{B}^{(3)} \prec_{\mathtt{CB}} \mathtt{C}^{(3)})$

$G\#\mathtt{B}\#\mathtt{c} \triangleq \ominus(\mathtt{B}^{(2)}); ! \cdot \Delta_3; \oplus(\mathtt{B}^{(3)}); \ominus(\mathtt{A}^{(1)} \prec_{\mathtt{HB}} \mathtt{B}^{(3)}); \oplus(\mathtt{C}^{(1)} \prec_{\mathtt{HB}} \mathtt{C}^{(3)})$

$G\#\mathtt{B}\#\mathtt{d} \triangleq ?\Delta_2 \cdot ; \oplus(\mathtt{B}^{(2)})$

$//\mathcal{C}(\mathtt{c},\mathtt{B},G\#\mathtt{B}\#\mathtt{c}) * \mathcal{C}(\mathtt{d},\mathtt{B},G\#\mathtt{B}\#\mathtt{d})$

```
    x = receive(d);
```

$//\ \mathcal{C}(\mathtt{c},\mathtt{B},G\#\mathtt{B}\#\mathtt{c}) * \mathcal{C}(\mathtt{d},\mathtt{B},\mathtt{emp}),\ \Pi := \Pi \cup \{\ominus(\mathtt{B}^{(2)})\}$

```
    send(c, … );
```

$//\ \mathcal{C}(\mathtt{c},\mathtt{B},\ominus(\mathtt{A}^{(1)} \prec_{\mathtt{HB}} \mathtt{B}^{(3)}); \oplus(\mathtt{C}^{(1)} \prec_{\mathtt{HB}} \mathtt{C}^{(3)})) * \mathcal{C}(\mathtt{d},\mathtt{B},\mathtt{emp}),\ \Pi := \ldots$

Example 3 – Verification

$$G(\mathtt{A},\mathtt{B},\mathtt{C},\mathtt{c},\mathtt{d}) \triangleq \mathtt{A}\xrightarrow{1}\mathtt{C} : \mathtt{c}\langle\Delta_1\rangle \; ; \; \mathtt{A}\xrightarrow{2}\mathtt{B} : \mathtt{d}\langle\Delta_2\rangle \; ; \; \mathtt{B}\xrightarrow{3}\mathtt{C} : \mathtt{c}\langle\Delta_3\rangle$$

$G\#\mathtt{All} \triangleq \oplus(\mathtt{A}^{(1)} \prec_{\mathtt{CB}} \mathtt{C}^{(1)}); \oplus(\mathtt{A}^{(1)} \prec_{\mathtt{HB}} \mathtt{A}^{(2)}); \oplus(\mathtt{A}^{(2)} \prec_{\mathtt{CB}} \mathtt{B}^{(2)}); \oplus(\mathtt{B}^{(2)} \prec_{\mathtt{HB}} \mathtt{B}^{(3)}); \oplus(\mathtt{C}^{(1)} \prec_{\mathtt{HB}} \mathtt{C}^{(3)}); \oplus(\mathtt{B}^{(3)} \prec_{\mathtt{CB}} \mathtt{C}^{(3)})$

$G\#\mathtt{B}\#\mathtt{c} \triangleq \ominus(\mathtt{B}^{(2)}); ! \cdot \Delta_3; \oplus(\mathtt{B}^{(3)}); \ominus(\mathtt{A}^{(1)} \prec_{\mathtt{HB}} \mathtt{B}^{(3)}); \oplus(\mathtt{C}^{(1)} \prec_{\mathtt{HB}} \mathtt{C}^{(3)})$

$G\#\mathtt{B}\#\mathtt{d} \triangleq ?\Delta_2 \cdot ; \oplus(\mathtt{B}^{(2)})$

$//\mathcal{C}(\mathtt{c},\mathtt{B}, G\#\mathtt{B}\#\mathtt{c}) * \mathcal{C}(\mathtt{d},\mathtt{B}, G\#\mathtt{B}\#\mathtt{d})$

```
    x = receive(d);
```

$// \ \mathcal{C}(\mathtt{c},\mathtt{B}, G\#\mathtt{B}\#\mathtt{c}) * \mathcal{C}(\mathtt{d},\mathtt{B},\mathtt{emp}), \ \Pi := \Pi \cup \{\ominus(\mathtt{B}^{(2)})\}$

```
    send(c, … );
```

$// \ \mathcal{C}(\mathtt{c},\mathtt{B}, \ominus(\mathtt{A}^{(1)} \prec_{\mathtt{HB}} \mathtt{B}^{(3)}); \oplus(\mathtt{C}^{(1)} \prec_{\mathtt{HB}} \mathtt{C}^{(3)})) * \mathcal{C}(\mathtt{d},\mathtt{B},\mathtt{emp}), \ \Pi := \ldots$

$// \ \mathcal{C}(\mathtt{c},\mathtt{B}, \ominus(\mathtt{A}^{(1)} \prec_{\mathtt{HB}} \mathtt{B}^{(3)}); \oplus(\mathtt{C}^{(1)} \prec_{\mathtt{HB}} \mathtt{C}^{(3)})) * \mathcal{C}(\mathtt{d},\mathtt{B},\mathtt{emp}), \ \Pi := \ldots \vdash \ \mathcal{C}(\mathtt{c},\mathtt{B},\mathtt{emp}) * \mathcal{C}(\mathtt{d},\mathtt{B},\mathtt{emp})$ FAIL

Communication Protocols – issues

Protocol

"A" sends a product id to "B" via channel "c"

Implementation

A	B
...	...
`send(c, "TV");`	`int x;`
	`x = receive(c);`

Communication Protocols – issues

Protocol

Type Safety

"A" sends a product id to "B" via channel "c"

Implementation

A

```
...
send(c, "TV");
```

B

```
...
int x;
x = receive(c);
```

Communication Protocols – issues

Protocol

Implementation

Type Safety

"A" sends a product id to "B" via channel "c"

A	B
...	...
`send(c, "TV");`	`int x;`
	`x = receive(c);`

"A" sends to "B" the number of required items via channel "d".

A	B
...	...
`send(d, 10);`	`x = receive(d);`
`send(d, 10);`	

Communication Protocols – issues

Protocol

Implementation

Type Safety

"A" sends a product id to "B" via channel "c"

A	B
…	…
send(c, "TV");	int x;
	x = receive(c);

Unexpected transmission

"A" sends to "B" the number of required items via channel "d".

A	B
…	…
send(d, 10);	x = receive(d);
send(d, 10);	

Communication Protocols – issues

Protocol

Implementation

Type Safety

"A" sends a product id to "B" via channel "c"

A	B
...	...
`send(c, "TV");`	`int x;`
	`x = receive(c);`

Unexpected transmission

"A" sends to "B" the number of required items via channel "d".

A	B
...	...
`send(d, 10);`	`x = receive(d);`
`send(d, 10);`	

"A" first sends the result to "B" and then to "C" via channel "c"

A	B	C
...	...	...
`send(c, "Pass");`	`a = receive(c);`	`a = receive(c);`
`send(c, "Fail");`		

Communication Protocols – issues

Protocol

Implementation

Type Safety

"A" sends a product id to "B" via channel "c"

A	B
…	…
`send(c, "TV");`	`int x;`
	`x = receive(c);`

Unexpected transmission

"A" sends to "B" the number of required items via channel "d".

A	B
…	…
`send(d, 10);`	`x = receive(d);`
`send(d, 10);`	

"A" first sends the result to "B" and then to "C" via channel "c"

A	B	C
…	…	…
`send(c, "Pass");`	`a = receive(c);`	`a = receive(c);`
`send(c, "Fail");`		

Who reads "Pass"?

Race on reading from c!

Communication Protocols – issues

Protocol

Implementation

Type Safety

"A" sends a product id to "B" via channel "c"

A

```
…
send(c, "TV");
```

B

```
…
int x;
x = receive(c);
```

Unexpected transmission

"A" sends to "B" the number of required items via channel "d".

A

```
…
send(d, 10);
send(d, 10);
```

B

```
…
x = receive(d);
```

Transmission Race

"A" first sends the result to "B" and then to "C" via channel "c"

A

```
…
send(c, "Pass");
send(c, "Fail");
```

B

```
…
a = receive(c);
```

C

```
…
a = receive(c);
```

Who reads "Pass"?

Race on reading from c!

Entailment Check – selected rules

$$\frac{\Delta_a \Rightarrow v_1 = v_2 \quad \mathcal{C}(v_1, P_1, L_a) \vdash \mathcal{C}(v_2, P_2, L_c) \rightsquigarrow S_1 \quad S_2 = \{\pi_i^e \mid \pi_i^e \in S_1 \text{ and } \mathtt{SAT}(\Delta_a * \Delta_c \wedge \pi_i^e)\} \quad \bigvee_{\pi^e \in S_2} (\Delta_a \wedge \pi^e) \vdash \Delta_c \rightsquigarrow S}{\mathcal{C}(v_1, P, L_a) * \Delta_a \vdash \mathcal{C}(v_2, P, L_c) * \Delta_c \rightsquigarrow S}\ \textbf{[ENT-CHAN-MATCH]}$$

$$\frac{S = \{\mathtt{emp} \wedge V = L_a\}}{L_a \vdash V \rightsquigarrow S}\ \textbf{[ENT-RHS-PVAR]}$$

$$\frac{P_1 = P_2 \quad L_a \vdash L_c \rightsquigarrow S' \quad S = \{\pi_i^e \mid \pi_i^e \in S'\}}{\mathcal{C}(v, P_1, L_a) \vdash \mathcal{C}(v, P_2, L_c) \rightsquigarrow S}\ \textbf{[ENT-CHAN]}$$

$$\frac{\Delta_a \vdash [v_1/v_2]\Delta_c \rightsquigarrow S' \quad S = \{\pi_i^e \mid \pi_i^e \in S'\}}{?v_1 \cdot \Delta_a \vdash ?v_2 \cdot \Delta_c \rightsquigarrow S}\ \textbf{[ENT-RECV]}$$

$$\frac{[v_1/v_2]\Delta_c \vdash \Delta_a \rightsquigarrow S' \quad S = \{\pi_i^e \mid \pi_i^e \in S'\}}{!v_1 \cdot \Delta_a \vdash !v_2 \cdot \Delta_c \rightsquigarrow S}\ \textbf{[ENT-SEND]}$$

$$\frac{\Box_a \vdash \Box_c \rightsquigarrow S_1 \quad L_a \vdash L_c \rightsquigarrow S_2 \quad \text{where } \Box ::= ?v \cdot \Delta \mid !v \cdot \Delta \mid f}{\Box_a ; L_a \vdash \Box_c ; L_c \rightsquigarrow \{\mathtt{emp} \wedge \pi_1 \wedge \pi_2 \mid \pi_1 \in S_1 \text{ and } \pi_2 \in S_2\}}\ \textbf{[ENT-SEQ]}$$

$$\frac{V \notin \mathrm{fv}(\Delta_c) \quad \mathtt{SAT}(\Delta_c) \quad \text{fresh } w \quad S = \{\mathtt{emp} \wedge V(w) = [w/v]\Delta_c\}}{V(v) \vdash \Delta_c \rightsquigarrow S}\ \textbf{[ENT-LHS-HO-VAR]}$$

$$\frac{V \notin \mathrm{fv}(\Delta_a) \quad \Delta_a \vdash \Delta_c \rightsquigarrow S' \quad \text{fresh } w \quad S = \{\mathtt{emp} \wedge V(w) = [w/v]\Delta_i \mid \Delta_i \in S'\}}{\Delta_a \vdash V(v) * \Delta_c \rightsquigarrow S}\ \textbf{[ENT-RHS-HO-VAR]}$$

$$\frac{L_i ; L_a \vdash L_c \rightsquigarrow S_i \quad S = \{\bigvee_i \Delta_i \mid \Delta_i \in S_i\}}{(\bigvee_i L_i); L_a \vdash L_c \rightsquigarrow S}\ \textbf{[ENT-LHS-OR]}$$

$$\frac{L_a \vdash L_i ; L_c \rightsquigarrow S_i \quad S = \bigcup S_i}{L_a \vdash (\bigvee_i L_i); L_c \rightsquigarrow S}\ \textbf{[ENT-RHS-OR]}$$

Entailment – extension of Concurrent Separation Logic

Separation Logic's frame rule:

$$\frac{\{\Phi_1\}\ C\ \{\Phi_2\}}{\{\Phi_1 * \Phi\}\ C\ \{\Phi_2 * \Phi\}}\quad \mathtt{fv}(\Phi) \cap \mathtt{modif}(\mathtt{C}) = \emptyset$$

CSL frame rule:

$$\frac{\{\Phi_1\}\ C\ \{\Phi_2\} \qquad \{\Phi_1'\}\ C'\ \{\Phi_2'\}}{\{\Phi_1 * \Phi_1'\}\ C \mid\mid C'\ \{\Phi_2 * \Phi_2'\}}\quad \begin{array}{l} (\mathtt{fv}(\Phi_1') \cup \mathtt{fv}(\Phi_2')) \cap \mathtt{modif}(\mathtt{C}) = \emptyset \\ (\mathtt{fv}(\Phi_1) \cup \mathtt{fv}(\Phi_2)) \cap \mathtt{modif}(\mathtt{C}') = \emptyset \end{array}$$

Entailment – extension of Concurrent Separation Logic

Separation Logic's frame rule:

$$\frac{\{\Phi_1\}\ C\ \{\Phi_2\}}{\{\Phi_1 * \Phi\}\ C\ \{\Phi_2 * \Phi\}}\quad \mathtt{fv}(\Phi) \cap \mathtt{modif}(\mathtt{C}) = \emptyset$$

CSL frame rule:

$$\frac{\{\Phi_1\}\ C\ \{\Phi_2\} \qquad \{\Phi_1'\}\ C'\ \{\Phi_2'\}}{\{\Phi_1 * \Phi_1'\}\ C\ ||\ C'\ \{\Phi_2 * \Phi_2'\}}\quad \begin{array}{l}(\mathtt{fv}(\Phi_1') \cup \mathtt{fv}(\Phi_2')) \cap \mathtt{modif}(\mathtt{C}) = \emptyset \\ (\mathtt{fv}(\Phi_1) \cup \mathtt{fv}(\Phi_2)) \cap \mathtt{modif}(\mathtt{C}') = \emptyset\end{array}$$

Separation in space!

Entailment – extension of Concurrent Separation Logic

Separation Logic's frame rule:

$$\frac{\{\Phi_1\}\ C\ \{\Phi_2\}}{\{\Phi_1 * \Phi\}\ C\ \{\Phi_2 * \Phi\}} \quad \mathtt{fv}(\Phi) \cap \mathtt{modif}(\mathtt{C}) = \emptyset$$

CSL frame rule:

$$\frac{\{\Phi_1\}\ C\ \{\Phi_2\} \qquad \{\Phi_1'\}\ C'\ \{\Phi_2'\}}{\{\Phi_1 * \Phi_1'\}\ C\ ||\ C'\ \{\Phi_2 * \Phi_2'\}}$$

Separation in space!

CSL + Ordering System: Separation in space + Separation in time

Orderings Collection

$$
\begin{array}{lll}
\textit{Border Base Element} & \mathtt{BForm\ a} & ::= \mathtt{a} \mid (\mathtt{BForm\ a}) * (\mathtt{BForm\ a}) \\
\textit{Border Element} & \mathtt{EForm\ a} & ::= \bot \mid \mathtt{BForm\ a} \mid (\mathtt{EForm\ a}) \vee (\mathtt{EForm\ a}) \\
\textit{Border Event} & \beta^{\mathtt{E}} & ::= \mathtt{EForm\ P}^{(i)} \\
\textit{Border Transmission} & \beta^{\mathtt{T}} & ::= \mathtt{EForm\ P} \xrightarrow{i} \mathtt{P} : \mathtt{c}
\end{array}
$$

$$
\textit{(Operation Map)}\ \mathtt{RMap} \stackrel{\text{def}}{=} \mathcal{R}\mathtt{ole} \to \beta^{\mathtt{E}} \quad \textit{(Transmission Map)}\ \mathtt{CMap} \stackrel{\text{def}}{=} \mathcal{C}\mathtt{han} \to \beta^{\mathtt{T}}
$$

$$
\textit{(Border)}\ \mathtt{Border} \stackrel{\text{def}}{=} \mathtt{RMap} \times \mathtt{CMap} \quad \textit{(Summary)}\ \mathtt{Summary} \stackrel{\text{def}}{=} \mathtt{Border} \times \mathtt{Border}
$$

Example 3:

$$
\boxed{\mathtt{A} \xrightarrow{1} \mathtt{C} : \mathtt{c}\ ;\ \mathtt{A} \xrightarrow{2} \mathtt{B} : \mathtt{d}\ ;\ \mathtt{B} \xrightarrow{3} \mathtt{C} : \mathtt{c}}
$$

Orderings Collection

$$
\begin{array}{llll}
\textit{Border Base Element} & \mathtt{BForm\ a} & ::= & \mathtt{a} \mid (\mathtt{BForm\ a}) * (\mathtt{BForm\ a}) \\
\textit{Border Element} & \mathtt{EForm\ a} & ::= & \bot \mid \mathtt{BForm\ a} \mid (\mathtt{EForm\ a}) \vee (\mathtt{EForm\ a}) \\
\textit{Border Event} & \beta^{\mathtt{E}} & ::= & \mathtt{EForm\ P}^{(i)} \\
\textit{Border Transmission} & \beta^{\mathtt{T}} & ::= & \mathtt{EForm\ P} \xrightarrow{i} \mathtt{P} : \mathtt{c}
\end{array}
$$

$$
\textit{(Operation Map)}\ \mathtt{RMap} \stackrel{\text{def}}{=} \mathcal{R}\mathtt{ole} \to \beta^{\mathtt{E}} \quad \textit{(Transmission Map)}\ \mathtt{CMap} \stackrel{\text{def}}{=} \mathcal{C}\mathtt{han} \to \beta^{\mathtt{T}}
$$

$$
\textit{(Border)}\ \mathtt{Border} \stackrel{\text{def}}{=} \mathtt{RMap} \times \mathtt{CMap} \quad \textit{(Summary)}\ \mathtt{Summary} \stackrel{\text{def}}{=} \mathtt{Border} \times \mathtt{Border}
$$

Example 3:

$$\boxed{\mathtt{A} \xrightarrow{1} \mathtt{C} : \mathtt{c}\ ;\ \mathtt{A} \xrightarrow{2} \mathtt{B} : \mathtt{d}\ ;\ \mathtt{B} \xrightarrow{3} \mathtt{C} : \mathtt{c}}$$

B F

B_1 F_1 B_2 F_2

$$\boxed{\mathtt{A} \xrightarrow{1} \mathtt{C} : \mathtt{c}}\ ;\ \boxed{\mathtt{A} \xrightarrow{2} \mathtt{B} : \mathtt{d}\ ;\ \mathtt{B} \xrightarrow{3} \mathtt{C} : \mathtt{c}}$$

Orderings Collection

$$
\begin{array}{llll}
\textit{Border Base Element} & \mathtt{BForm\ a} & ::= & \mathtt{a} \mid (\mathtt{BForm\ a}) * (\mathtt{BForm\ a}) \\
\textit{Border Element} & \mathtt{EForm\ a} & ::= & \bot \mid \mathtt{BForm\ a} \mid (\mathtt{EForm\ a}) \vee (\mathtt{EForm\ a}) \\
\textit{Border Event} & \beta^{\mathtt{E}} & ::= & \mathtt{EForm\ P}^{(i)} \\
\textit{Border Transmission} & \beta^{\mathtt{T}} & ::= & \mathtt{EForm\ P} \xrightarrow{i} \mathtt{P : c}
\end{array}
$$

$$
\textit{(Operation Map)}\ \mathtt{RMap} \stackrel{\text{def}}{=} \mathcal{R}\mathtt{ole} \to \beta^{\mathtt{E}} \qquad \textit{(Transmission Map)}\ \mathtt{CMap} \stackrel{\text{def}}{=} \mathcal{C}\mathtt{han} \to \beta^{\mathtt{T}}
$$

$$
\textit{(Border)}\ \mathtt{Border} \stackrel{\text{def}}{=} \mathtt{RMap} \times \mathtt{CMap} \qquad \textit{(Summary)}\ \mathtt{Summary} \stackrel{\text{def}}{=} \mathtt{Border} \times \mathtt{Border}
$$

Example 3:

$$\boxed{\mathtt{A} \xrightarrow{1} \mathtt{C : c}\ ;\ \mathtt{A} \xrightarrow{2} \mathtt{B : d}\ ;\ \mathtt{B} \xrightarrow{3} \mathtt{C : c}}$$

$\{\mathtt{A}^{(1)}, \mathtt{B}^{(2)}, \mathtt{C}^{(1)}\}$ $\{\mathtt{d}^{(2)}, \mathtt{c}^{(1)}\}$ B

$\{\mathtt{A}^{(2)}, \mathtt{B}^{(3)}, \mathtt{C}^{(3)}\}$ $\{\mathtt{d}^{(2)}, \mathtt{c}^{(3)}\}$ F

$\mathtt{B} := \mathtt{merge}(\mathtt{B}_1, \mathtt{B}_2)$

$\mathtt{F} := \mathtt{merge}(\mathtt{F}_2, \mathtt{F}_1)$

B_1: $\{\mathtt{A}^{(1)}, \mathtt{C}^{(1)}\}$ $\{\mathtt{c}^{(1)}\}$

F_1: $\{\mathtt{A}^{(1)}, \mathtt{C}^{(1)}\}$ $\{\mathtt{c}^{(1)}\}$

B_2: $\{\mathtt{A}^{(2)}, \mathtt{B}^{(2)}, \mathtt{C}^{(3)}\}$ $\{\mathtt{d}^{(2)}, \mathtt{c}^{(3)}\}$

F_2: $\{\mathtt{A}^{(2)}, \mathtt{B}^{(3)}, \mathtt{C}^{(3)}\}$ $\{\mathtt{d}^{(2)}, \mathtt{c}^{(3)}\}$

$$\boxed{\mathtt{A} \xrightarrow{1} \mathtt{C : c}}\ ;\ \boxed{\mathtt{A} \xrightarrow{2} \mathtt{B : d}\ ;\ \mathtt{B} \xrightarrow{3} \mathtt{C : c}}$$

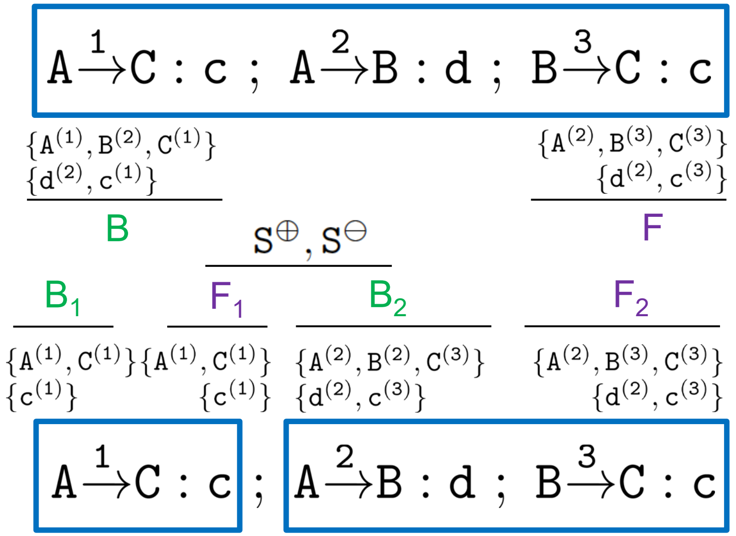
Orderings Collection

$$
\begin{array}{llll}
\textit{Border Base Element} & \mathtt{BForm\ a} & ::= & \mathtt{a} \mid (\mathtt{BForm\ a}) * (\mathtt{BForm\ a}) \\
\textit{Border Element} & \mathtt{EForm\ a} & ::= & \bot \mid \mathtt{BForm\ a} \mid (\mathtt{EForm\ a}) \vee (\mathtt{EForm\ a}) \\
\textit{Border Event} & \beta^{\mathtt{E}} & ::= & \mathtt{EForm\ P}^{(i)} \\
\textit{Border Transmission} & \beta^{\mathtt{T}} & ::= & \mathtt{EForm\ P} \xrightarrow{i} \mathtt{P : c}
\end{array}
$$

$$
\textit{(Operation Map)}\ \mathtt{RMap} \stackrel{\text{def}}{=} \mathcal{R}\mathtt{ole} \to \beta^{\mathtt{E}} \quad \textit{(Transmission Map)}\ \mathtt{CMap} \stackrel{\text{def}}{=} \mathcal{C}\mathtt{han} \to \beta^{\mathtt{T}}
$$

$$
\textit{(Border)}\ \mathtt{Border} \stackrel{\text{def}}{=} \mathtt{RMap} \times \mathtt{CMap} \quad \textit{(Summary)}\ \mathtt{Summary} \stackrel{\text{def}}{=} \mathtt{Border} \times \mathtt{Border}
$$

Example 3:



$$
\begin{aligned}
\mathtt{B} &:= \mathtt{merge}(\mathtt{B}_1, \mathtt{B}_2) \\
\mathtt{F} &:= \mathtt{merge}(\mathtt{F}_2, \mathtt{F}_1)
\end{aligned}
$$

$$
\begin{aligned}
\mathtt{S}^{\oplus} &:= \mathtt{S}^{\oplus} \cup \{\mathtt{A}^{(1)} \prec_{\mathtt{HB}} \mathtt{A}^{(2)}, \mathtt{C}^{(1)} \prec_{\mathtt{HB}} \mathtt{C}^{(3)}\} \\
\mathtt{S}^{\ominus} &:= \mathtt{S}^{\ominus} \cup \{1 \prec_{\mathtt{HB}} 3\}
\end{aligned}
$$

Well-formedness (*)

[**Well-Formed Concurrency**] A protocol specification, $G_1 * G_2$, is said to be well-formed with respect to $*$ if and only if $\forall \mathsf{c} \in G_1 \implies \mathsf{c} \notin G_2$, and vice versa.

Well-formedness (⋁)

(a) *(same first channel)* $\forall \mathtt{c}_1 \in i_{\mathtt{k}}, \mathtt{c}_2 \in \mathtt{l}_{\mathtt{j}} \Rightarrow \mathtt{c}_1 = \mathtt{c}_2$;

(b) *(same first sender* $\mathtt{S}$*)* $\forall \mathtt{S}_1 \in i_{\mathtt{k}}, \mathtt{S}_2 \in \mathtt{l}_{\mathtt{j}} \Rightarrow \mathtt{S}_1 = \mathtt{S}_2 \wedge \mathtt{S} = \mathtt{S}_1$;

(c) *(same first receiver* $\mathtt{R}$*)* $\forall \mathtt{R}_1 \in i_{\mathtt{k}}, \mathtt{R}_2 \in \mathtt{l}_{\mathtt{j}} \Rightarrow \mathtt{R}_1 = \mathtt{R}_2 \wedge \mathtt{R} = \mathtt{R}_1$;

(d) *(mutually exclusive "first" messages)*

$$\forall \mathtt{j}, \mathtt{k} \in \{i_1, .., i_{\mathtt{n}}, \mathtt{l}_1, .., \mathtt{l}_{\mathtt{m}}\} \Rightarrow \mathtt{UNSAT}(\Delta_{\mathtt{j}} \wedge \Delta_{\mathtt{k}}) \vee \mathtt{j} = \mathtt{k};$$

(e) *(same roles)* $\forall \mathtt{P} \in \mathtt{G}_1 \vee \mathtt{G}_2 \Rightarrow \mathtt{P} = \mathtt{S} \vee \mathtt{P} = \mathtt{R}$, *with peers* $\mathtt{S}$ *and* $\mathtt{R}$ *the roles referenced by conditions (b) and (c), respectively;*

(f) *(recursive well-formedness)* $\mathtt{G}_1$ *and* $\mathtt{G}_2$ *are well-formed with respect to* $\vee$.

A Session Logic for
Relaxed Communication Protocols

Relaxed Communication Protocols – Motivation (i)

“A” first sends the result to “B” and then to “C” via channel “c”

(1)

A	B	C
…	…	…
`send(c, “Pass”);`	`a = receive(c);`	`a = receive(c);`
`send(c, “Fail”);`		

Relaxed Communication Protocols – Motivation (i)

"A" first sends the result to "B" and then to "C" via channel "c"

(1)

A	B	C

`…`
`send(c, "Pass");`
`send(c, "Fail");`
	`…`
	`a = receive(c);`
		`…`
		`a = receive(c);`

(2)

A	B	C

`…`
`send(c, "Pass");`
`send(c, "Fail");`
	`…`
	`a = receive(c);`
	`notifyAll(w);`
		`…`
		`wait(w);`
		`a = receive(c);`

Relaxed Communication Protocols – Motivation (i)

"A" first sends the result to "B" and then to "C" via channel "c"

(1)

A	B	C

`...`
`send(c, "Pass");`
`send(c, "Fail");`
	`...`
	`a = receive(c);`
		`...`
		`a = receive(c);`

(2)

A	B	C

`...`
`send(c, "Pass");`
`send(c, "Fail");`
	`...`
	`a = receive(c);`
	`notifyAll(w);`
		`...`
		`wait(w);`
		`a = receive(c);`

Current approaches for session formalization declare this protocol as UNSAFE!
(due to race on reading from "c")

Relaxed Communication Protocols – Motivation (i)

"A" first sends the result to "B" and then to "C" via channel "c"

(1)

A	B	C

`…`
`send(c, "Pass");`
`send(c, "Fail");`
	`…`
	`a = receive(c);`
		`…`
		`a = receive(c);`

(2)

A	B	C

`…`
`send(c, "Pass");`
`send(c, "Fail");`
	`…`
	`a = receive(c);`
	`notifyAll(w);`
		`…`
		`wait(w);`
		`a = receive(c);`

Current approaches for session formalization declare this protocol as UNSAFE!
(due to race on reading from "c")

Our goal: relax the tag of "SAFE" protocols, and enforce safety at the program code level.

Relaxed Communication Protocols – Motivation (ii)

“B” and “C” send their computation result to “A” via channel “c”

A		B		C
…	‖	…	‖	…
`x = receive(c);`	‖	`send(c,10);`	‖	`send(c,15);`
`y = receive(c);`	‖		‖	
`return x + y;`				

Relaxed Communication Protocols – Motivation (ii)

"B" and "C" send their computation result to "A" via channel "c"

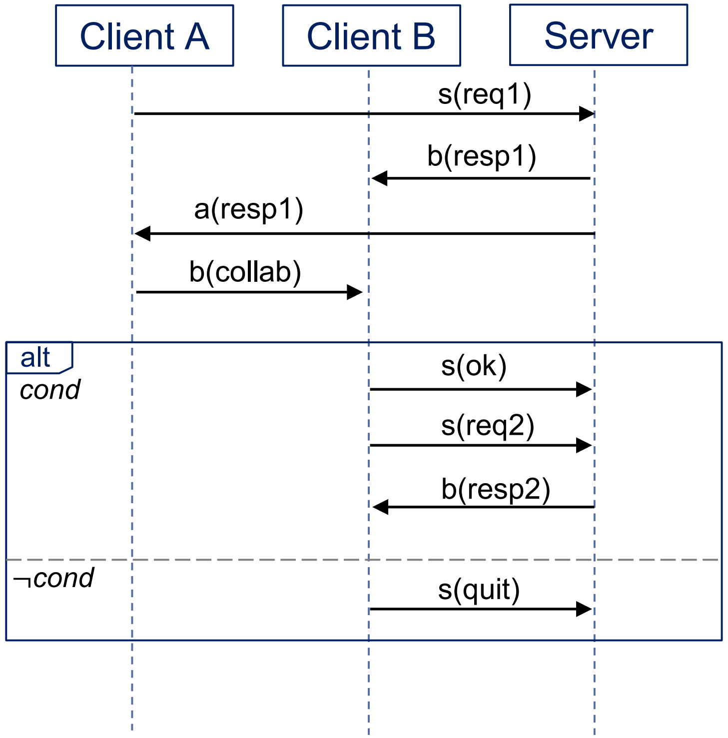
A		B		C

`…`
`x = receive(c);`
`y = receive(c);`
`return x + y;`
	‖	`…`
		`send(c,10);`
			‖	`…`
				`send(c,15);`

Current approaches for session formalization declare this protocol as UNSAFE!
(due to race on sending to "c")

However, parallel computing has been used to model difficult problems in many areas: rush hour traffic, weather, auto assembly, photonics, molecular sciences, etc.

Collaborative Client – Server (revisited)



$$
\begin{array}{lll}
\textit{Global protocol} & G & ::= \\
\textit{Single transmission} & & \mathtt{S}\xrightarrow{i}\mathtt{R} : \mathtt{c}\langle \mathtt{v}\cdot\Delta\rangle \\
\textit{Concurrency} & & \mid\ G * G \\
\textit{Choice} & & \mid\ G \vee G \\
\textit{Sequencing} & & \mid\ G\ ;\ G \\
\textit{Inaction} & & \mid\ \mathtt{emp}
\end{array}
$$

$$
\begin{array}{lll}
G_{\mathtt{ABS}} & \triangleq & \mathtt{A}\xrightarrow{1}\mathtt{S} : \mathtt{s}\langle \mathtt{String}\rangle\ ; \\
& & (\mathtt{S}\xrightarrow{2}\mathtt{B} : \mathtt{b}\langle \mathtt{v}\cdot \mathtt{v} > 0\rangle * \mathtt{S}\xrightarrow{3}\mathtt{A} : \mathtt{a}\langle \mathtt{v}\cdot \mathtt{v} > 0\rangle)\ ; \\
& & \mathtt{A}\xrightarrow{4}\mathtt{B} : \mathtt{b}\langle \mathtt{v}\cdot \mathtt{v} \geq 0\rangle\ ; \\
& & (\mathtt{B}\xrightarrow{5}\mathtt{S} : \mathtt{s}\langle \mathtt{ok}\rangle\ ;\ \mathtt{B}\xrightarrow{6}\mathtt{S} : \mathtt{s}\langle \mathtt{v}\cdot \mathtt{Addr(v)}\rangle\ ;\ \mathtt{S}\xrightarrow{7}\mathtt{B} : \mathtt{b}\langle \mathtt{v}\cdot \mathtt{Date(v)}\rangle \\
& & \vee\, \mathtt{B}\xrightarrow{8}\mathtt{S} : \mathtt{s}\langle \mathtt{quit}\rangle).
\end{array}
$$

Take – away 1: TYPE SYSTEMS -> LOGIC

Example 1

"A" first sends the result to "B" and then to "C" via channel "c"

$$\mathtt{A} \xrightarrow{1} \mathtt{B} : \mathtt{c}\langle \text{"}\mathtt{Pass}\text{"} \rangle ; \mathtt{A} \xrightarrow{2} \mathtt{C} : \mathtt{c}\langle \text{"}\mathtt{Fail}\text{"} \rangle$$

A	B	C
…	…	…
`send(c, "Pass");`	`a = receive(c);`	
`send(c, "Fail");`	`notifyAll(w);`	`wait(w);`
		`a = receive(c);`

Example 1

“A” first sends the result to “B” and then to “C” via channel “c”

$$\mathtt{A}\xrightarrow{1}\mathtt{B} : \mathtt{c}\langle\text{"}\mathtt{Pass}\text{"}\rangle ; \mathtt{A}\xrightarrow{2}\mathtt{C} : \mathtt{c}\langle\text{"}\mathtt{Fail}\text{"}\rangle$$

A		B		C
…	∥	…	∥	…
`send(c, "Pass");`		`a = receive(c);`		
`send(c, "Fail");`		`notifyAll(w);`		`wait(w);`
				`a = receive(c);`

Example 1

"A" first sends the result to "B" and then to "C" via channel "c"

$$\mathtt{A} \xrightarrow{1} \mathtt{B} : \mathtt{c}\langle\mathtt{"Pass"}\rangle ; \mathtt{A} \xrightarrow{2} \mathtt{C} : \mathtt{c}\langle\mathtt{"Fail"}\rangle$$

A		B		C
…	‖	…	‖	…
`send(c, "Pass");`	‖	`a = receive(c);`	‖	
`send(c, "Fail");`	‖	`notifyAll(w);`	‖	`wait(w);`
				`a = receive(c);`

Introduce a proof obligation on event ordering to prove that
B ***happens-before*** C

Example 2

"A" sends to "B" a string and then "C" sends to "B" an integer via channel "c"

$$\mathtt{A}\xrightarrow{1}\mathtt{B} : \mathtt{c}\langle\mathtt{String}\rangle ; \mathtt{C}\xrightarrow{2}\mathtt{B} : \mathtt{c}\langle\mathtt{int}\rangle$$

A		B		C
…	‖	…	‖	…
`send(c, "TV");`	‖	`String x = receive(c);`	‖	
	‖	`int    y = receive(c);`	‖	
				`send(c,2);`

Race on writing to c!

Example 2

“A” sends to “B” a string and then “C” sends to “B” an integer via channel “c”

$$\mathtt{A} \xrightarrow{1} \mathtt{B} : \mathtt{c}\langle \mathtt{String} \rangle ; \mathtt{C} \xrightarrow{2} \mathtt{B} : \mathtt{c}\langle \mathtt{int} \rangle$$

A	B	C
…	…	…
`send(c, “TV”);`	`String x = receive(c);`	
`notifyAll(w);`	`int    y = receive(c);`	`wait(w);`
		`send(c,2);`

Example 2

"A" sends to "B" a string and then "C" sends to "B" an integer via channel "c"

$$\mathtt{A} \xrightarrow{1} \mathtt{B} : \mathtt{c}\langle \mathtt{String} \rangle ; \mathtt{C} \xrightarrow{2} \mathtt{B} : \mathtt{c}\langle \mathtt{int} \rangle$$

A		B		C
…	‖	…	‖	…
`send(c, "TV");`	‖	`String x = receive(c);`	‖	
`notifyAll(w);`	‖	`int    y = receive(c);`	‖	`wait(w);`
				`send(c,2);`

Introduce a proof obligation on event ordering to prove that
A ***happens-before*** C

Introduce Race-Free Guards

$$\mathtt{S}_1 \xrightarrow{\mathtt{i}_1} \mathtt{R}_1 : \mathtt{c}\langle \Delta_1 \rangle ; \mathtt{S}_2 \xrightarrow{\mathtt{i}_2} \mathtt{R}_2 : \mathtt{c}\langle \Delta_2 \rangle$$

To ensure race-freedom on c, prove that:

$$\mathtt{S}_1^{(i_1)} \prec_{\mathtt{HB}} \mathtt{S}_2^{(i_2)} \;\wedge\; \mathtt{R}_1^{(i_1)} \prec_{\mathtt{HB}} \mathtt{R}_2^{(i_2)} \qquad \Leftrightarrow \qquad i_1 \prec_{\mathtt{HB}} i_2$$

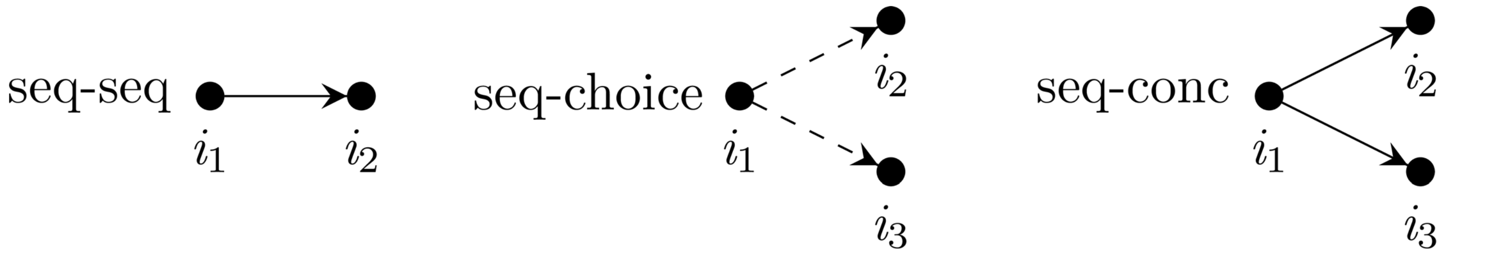
HB between events — HB between transmissions

Happens-Before Relation

Definition 1 (Happens-before) *Given a global protocol* $\mathtt{G}$*, two events* $\mathtt{P}_1^{(i_1)}$ *and* $\mathtt{P}_2^{(i_2)}$ *are said to be in a happens-before relation in* $\mathtt{G}$*,* $\mathtt{P}_1^{(i_1)} \prec_{\mathtt{HB}} \mathtt{P}_2^{(i_2)}$*, if and only if* $\mathtt{P}_1^{(i_1)}$ *completes prior to* $\mathtt{P}_2^{(i_2)}$*,* $i_1 \neq i_2$*.*

1. **Transitive:** $\mathtt{P}_1^{(i_1)} \prec_{\mathtt{HB}} \mathtt{P}_2^{(i_2)} \wedge \mathtt{P}_2^{(i_2)} \prec_{\mathtt{HB}} \mathtt{P}_3^{(i_3)} \Rightarrow \mathtt{P}_1^{(i_1)} \prec_{\mathtt{HB}} \mathtt{P}_3^{(i_3)}$
2. **Irreflexive:** $\forall \mathtt{P}_1, \mathtt{P}_2, i_1, i_2 \in G \cdot \mathtt{P}_1^{(i_1)} \prec_{\mathtt{HB}} \mathtt{P}_2^{(i_2)} \Rightarrow i_1 \neq i_2$
3. **Asymmetric:** $\forall \mathtt{P}_1, \mathtt{P}_2, i_1, i_2 \in G \cdot \mathtt{P}_1^{(i_1)} \prec_{\mathtt{HB}} \mathtt{P}_2^{(i_2)} \Rightarrow \neg(\mathtt{P}_2^{(i_2)} \prec_{\mathtt{HB}} \mathtt{P}_1^{(i_1)})$

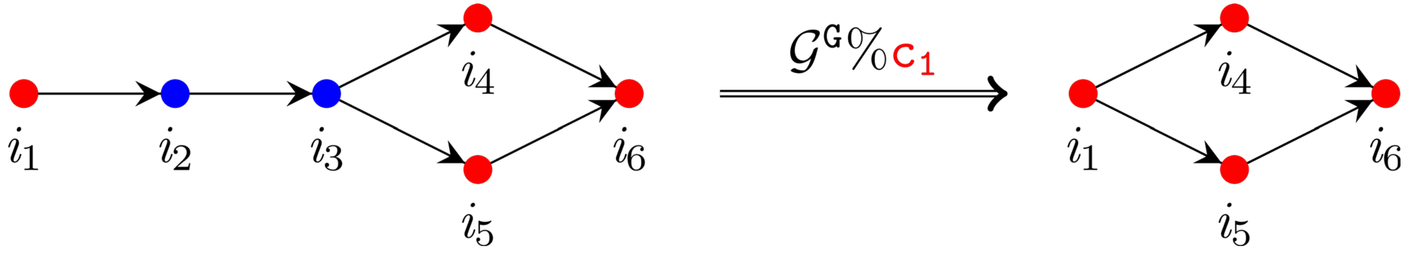
Protocols Diagrammatic View



$(i_1; i_2)$ $\quad$ $(i_1; (i_2 \vee i_3))$ $\quad$ $(i_1; (i_2 * i_3))$

Example to highlight adjacent transmissions:

$$G \triangleq \mathtt{A} \xrightarrow{i_1} \mathtt{C} : \mathtt{c}_1 \,;\, \mathtt{B} \xrightarrow{i_2} \mathtt{C} : \mathtt{c}_2 \,;\, \mathtt{A} \xrightarrow{i_3} \mathtt{C} : \mathtt{c}_2 \,;\, (\mathtt{A} \xrightarrow{i_4} \mathtt{B} : \mathtt{c}_1 * \mathtt{A} \xrightarrow{i_5} \mathtt{B} : \mathtt{c}_1) \,;\, \mathtt{A} \xrightarrow{i_6} \mathtt{C} : \mathtt{c}_1 .$$



$$\|i_1; i_4\|_G^{\mathtt{c}_1} \;,\; \|i_1; i_5\|_G^{\mathtt{c}_1} \;,\; \|i_4; i_6\|_G^{\mathtt{c}_1} \;,\; \|i_5; i_6\|_G^{\mathtt{c}_1}$$

COMMUNICATION PROTOCOLS – issues (revisited)

Protocol

Type Safety

"A" sends a product id to "B" via channel "c"

$$G(\mathtt{A},\mathtt{B},\mathtt{c}) \triangleq \mathtt{A} \xrightarrow{1} \mathtt{B} : \mathtt{c}\langle \mathtt{String} \rangle .$$

A		B
…	||	…
`send(c, "TV");`	||	`int x;`
	||	`x = receive(c);`

FAIL

COMMUNICATION PROTOCOLS – issues (revisited)

Protocol

Type Safety

"A" sends a product id to "B" via channel "c"

$G(\mathtt{A},\mathtt{B},\mathtt{c}) \triangleq \mathtt{A} \xrightarrow{1} \mathtt{B} : \mathtt{c}\langle \mathtt{String} \rangle.$

A		B
…	‖	…
`send(c, "TV");`	‖	`int x;`
	‖	`x = receive(c);`

FAIL

Verification fails due to unexpected transmission

"A" sends to "B" the number of required items via channel "d".

$G(\mathtt{A},\mathtt{B},\mathtt{d}) \triangleq \mathtt{A} \xrightarrow{1} \mathtt{B} : \mathtt{d}\langle \mathtt{int} \rangle. \implies \mathcal{C}(\mathtt{d},\mathtt{A},!\mathtt{int};\oplus(\mathtt{A}^{(1)}))$

A		B
…	‖	…
`x = receive(d);`	‖	`x = receive(d);`
`send(d, 10);`	‖	

COMMUNICATION PROTOCOLS – issues (revisited)

"A" first sends the result to "B" and then to "C" via channel "c"

$$G(\mathtt{A},\mathtt{B},\mathtt{C},\mathtt{c}) \triangleq \mathtt{A}\xrightarrow{1}\mathtt{B} : \mathtt{c}\langle\text{"}\mathtt{Fail}\text{"}\rangle ; \mathtt{A}\xrightarrow{2}\mathtt{C} : \mathtt{c}\langle\text{"}\mathtt{Pass}\text{"}\rangle \longrightarrow \ominus(\mathtt{B}^{(1)} \prec_{\mathtt{HB}} \mathtt{C}^{(2)})$$

Fail due to data race

A	B	C
…	…	…
`send(c, "Pass");`	`a = receive(c);`	`a = receive(c);`
`send(c, "Fail");`		

Succeeds due to explicit sync

A	B	C
…	…	…
`send(c, "Yes");`	`a = receive(c);`	
`send(c, "No");`	`notifyAll(w);`	`wait(w);`
		`a = receive(c);`

Relaxed Communication Protocols - issues (revisited)

Nondeterminism: $G(\mathtt{A},\mathtt{B},\mathtt{C},\mathtt{c}) \triangleq \mathtt{B} \xrightarrow{1} \mathtt{A} : \mathtt{c}\langle\mathtt{int}\rangle * \mathtt{C} \xrightarrow{2} \mathtt{A} : \mathtt{c}\langle\mathtt{int}\rangle.$

Succeeds with extra conditions: (i) same receiver, (ii) equivalent messages

$\mathtt{R}^{(1)} = \mathtt{R}^{(2)}$ $\Delta_1 \dashv\vdash \Delta_2$

A

```
...
x = receive(c);
y = receive(c);
return x + y;
```

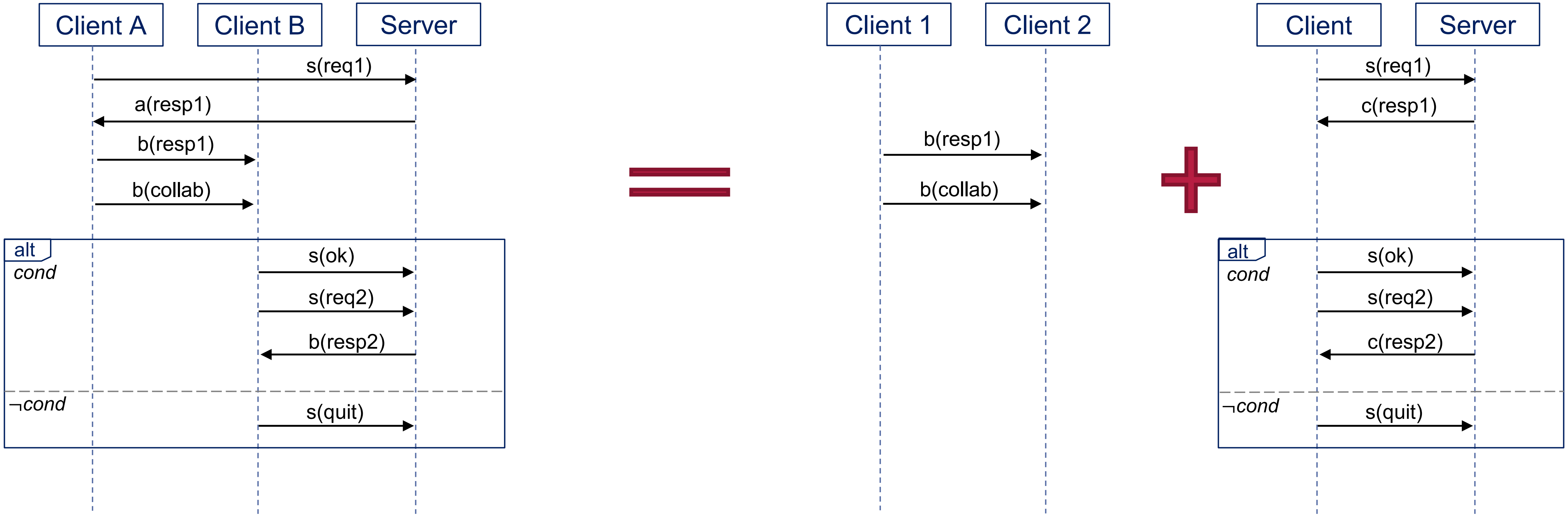
B

```
...
send(c,10);
```

C

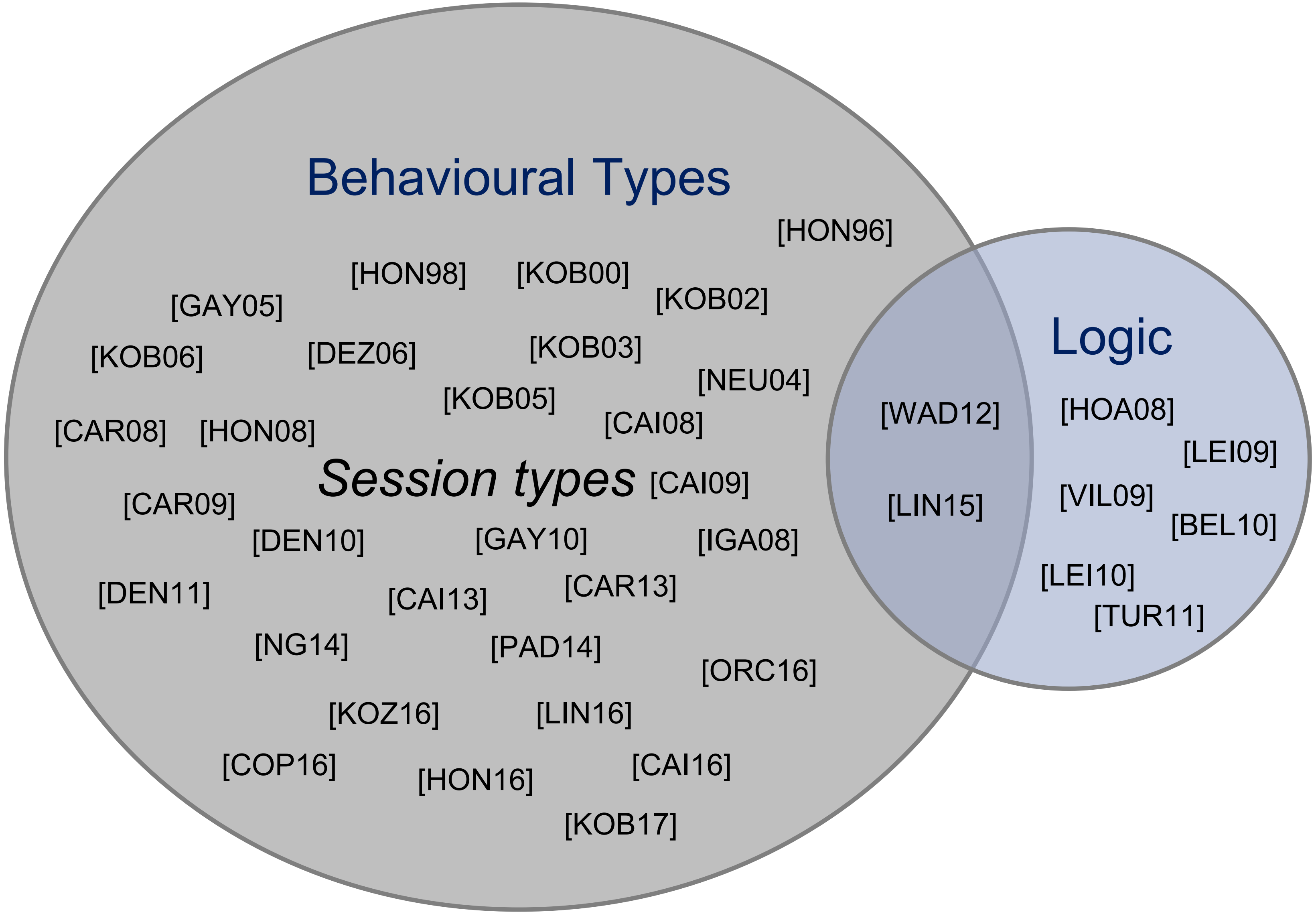
```
...
send(c,15);
```

Modular Protocols



1. Make protocols instantiable by adding protocol parameters.
2. Attach a labelling system which contains instantiable labels and maintains uniqueness of transmissions.
3. Create event ordering summaries for each predicate.

State of the Art



State of the Art

BEHAVIORAL TYPES [HONDA, POPL'96]
[KOBAYASHI, IC'02] [KOBAYASHI, TCS'00]
[KOBAYASHI, LNCS'03] [CAIRES, TCS'08] [KOBAYASHI, AI'05]
[KOBAYASHI, CONCUR'06] [KOBAYASHI et al, IC'07]
[IGARASHI and KOBAYASHI, TCS'04] [CAIRES and SECO, 2013]

PROGRAM LOGICS FOR CONCURRENCY
[O'HEARN, CONCUR'04]

SESSION TYPES [HONDA et al., ESOP'98]
[NEUBAUER et al, PADL'04] [GAY et al., AI'05] [GAY et al., JFP'10]
[HONDA et al., POPL'08] [CARBONE et al., CT'08]
[DENIÉLOU and YOSHIDA et al., POPL'11]
[CARBONE et al., POPL'13] [CAIRES and VIEIRA, ESOP'09]
[ORCHARD and YOSHIDA et al., POPL'16]
[KOUZAPAS et al., MSCS'16] [COPPO et al., MSCS'16]
[CAPECCHI et al., MSCS'16] [CARBONE, TCS'09]
[LÓPEZ et al., OOPSLA'15] [BOCCHI, CONCUR'10]
PADDLE
[NG and YOSHIDA, PDP'15] [LANGE et al., POPL'17]
[HU and YOSHIDA, FASE'17] [HU and YOSHIDA, FASE'16]
SCRIBBLE
[LANGE and YOSHIDA, FASE'17] [YOSHIDA et al., TGC'13]

[CAIRES and PFENNING, CONCUR'10]
[CAIRES et al., MSCS'12]
[WADLER, ICFP'12]
[CARBONE et al., CONCUR'15]
[LINDLEY and MORRIS, ESOP'15]
[CAIRES and LOPEZ, FORTE'16]
[CARBONE et al., CONCUR'16]
[CARBONE et al., AI'17]

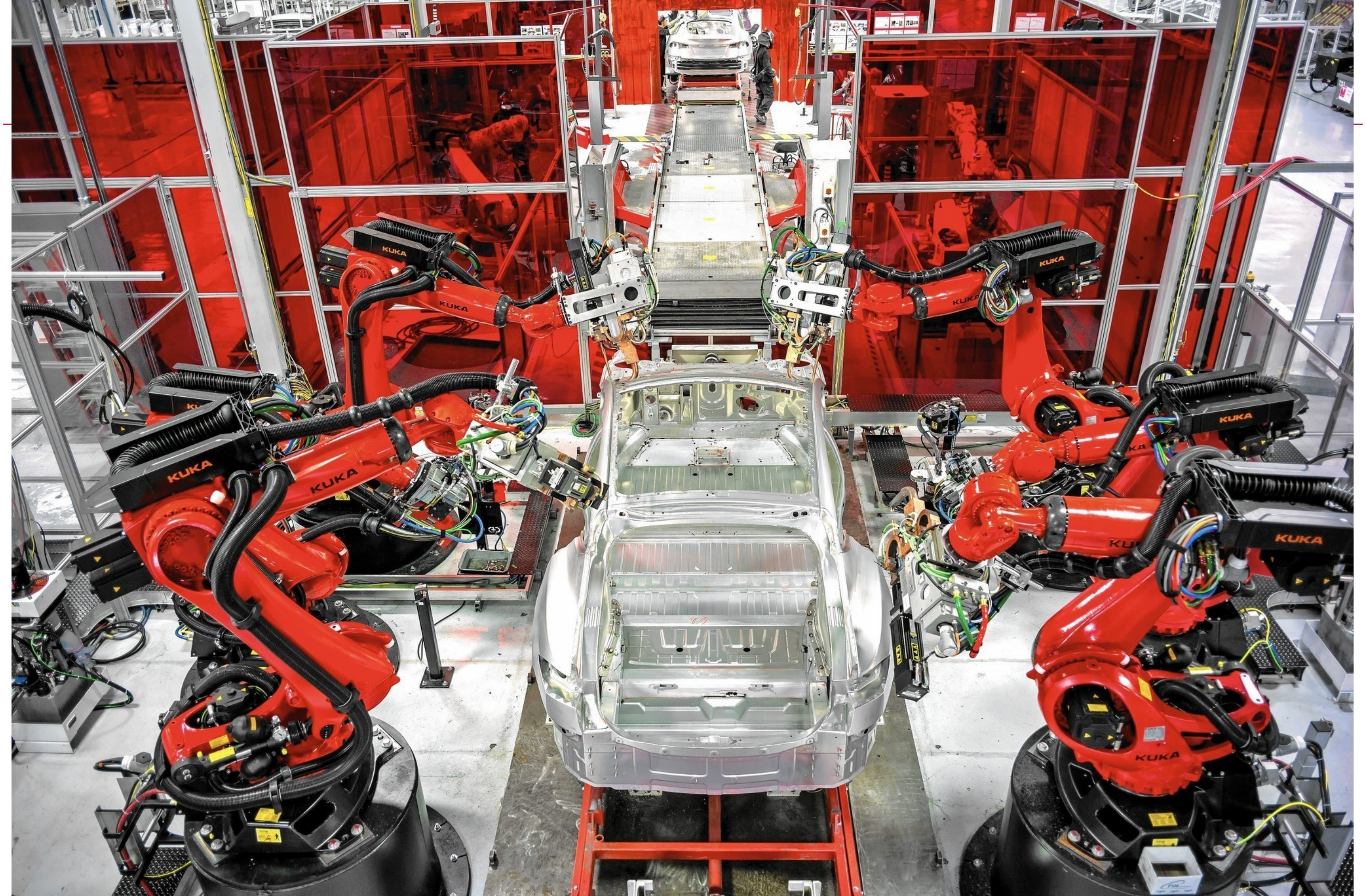
PROVING PROTOCOLS

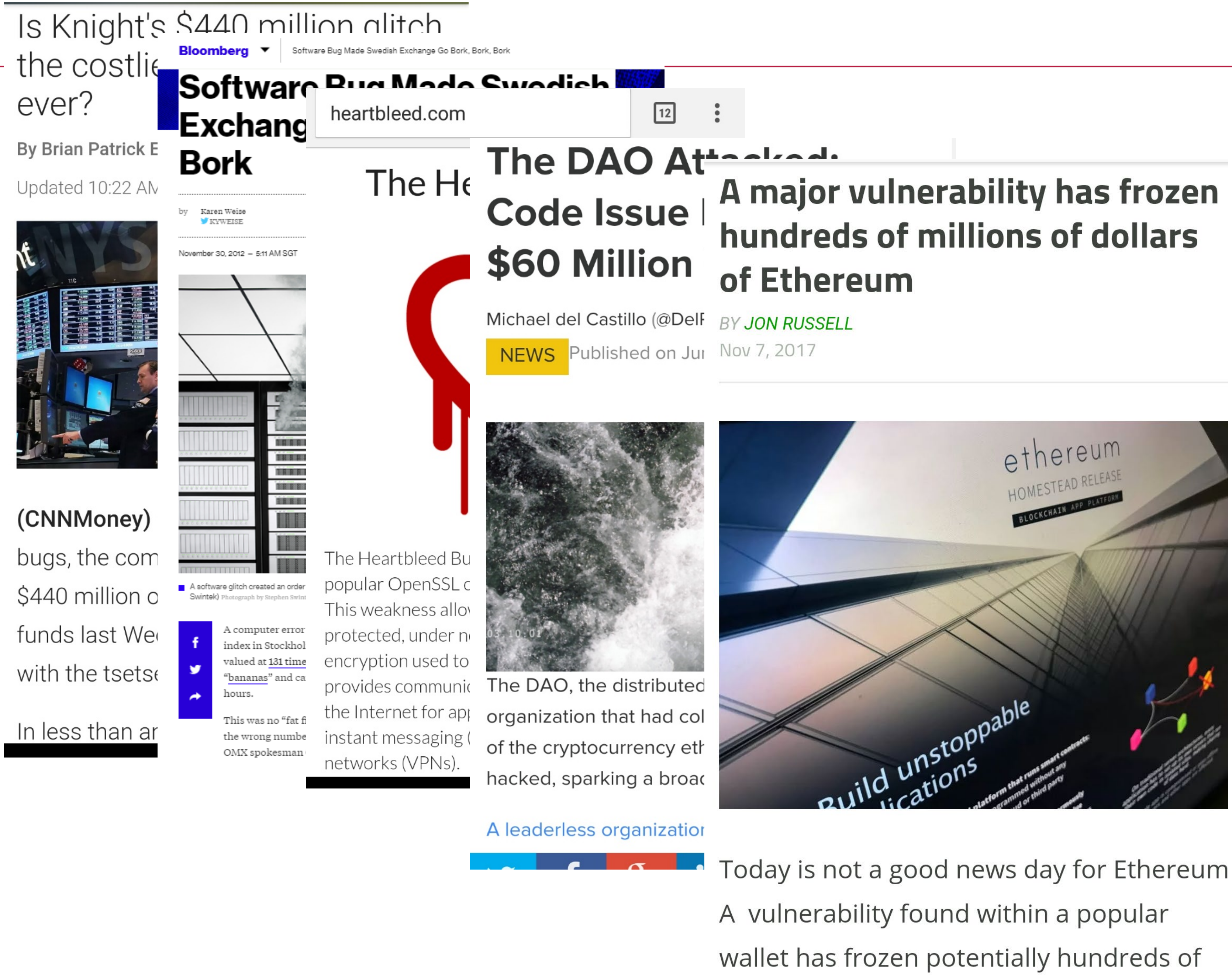
Related Work

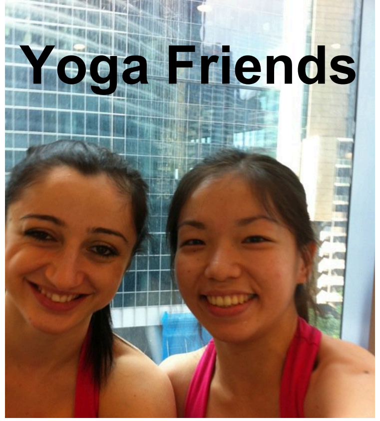
Logics with channel primitives:

- CSL for copyless message passing [VIL09]: an extension of separation for bidirectional communication between two players using global contracts
- CSL for pipelined parallelization [BEL10]: an extension of separation logic which supports multiple players communicating through a single shared channel
- Chalice[LEI09] with support for message passing [LEI10]: modular verification to prevent deadlocks of programs which mix message passing and locking.

[VIL09] VILLARD , J., L OZES , É., and C ALCAGNO , C., “Proving copyless message passing,” in APLAS 2009 , pp. 194–209, Springer.
[BEL10] BELL , C. J., APPEL , A. W., and WALKER , D., “Concurrent Separation Logic for Pipelined Parallelization,” in SAS 2010, pp. 151–166, Springer.
[LEI10] LEINO , K. R. M., MÜLLER , P., and SMANS , J., “Deadlock-Free Channels and Locks,” in ESOP 2010, pp. 407–426, Springer.
[LEI09] LEINO , K. R. M. and MÜLLER , P., “A Basis for Verifying Multi-Threaded Programs,” in ESOP 2009 pp. 378–393, Springer.


KUKA


Is Knight's $440 million glitch
the costli
ever?
By Brian Patrick E
Updated 10:22 AM
(CNNMoney)
bugs, the com
$440 million o
funds last We
with the tsets
In less than ar
Bloomberg
Software Bug Made Swedish Exchange Go Bork, Bork, Bork
Software Bug Made Swedish
Exchang
Bork
by Karen Weise
KYWEISE
November 30, 2012 – 5:11 AM SGT
A software glitch created an order
Swintek)
A computer error
index in Stockhol
valued at 131 time
"bananas" and ca
hours.
This was no "fat f
the wrong numbe
OMX spokesman
heartbleed.com
12
The He
The Heartbleed Bu
popular OpenSSL c
This weakness allo
protected, under n
encryption used to
provides communi
the Internet for ap
instant messaging (
networks (VPNs).
The DAO Attacked:
Code Issue
$60 Million
Michael del Castillo (@DelF
NEWS
Published on Ju
The DAO, the distributed
organization that had col
of the cryptocurrency eth
hacked, sparking a broa
A leaderless organizatio
A major vulnerability has frozen hundreds of millions of dollars of Ethereum
BY JON RUSSELL
Nov 7, 2017
ethereum
HOMESTEAD RELEASE
BLOCKCHAIN APP PLATFORM
Build unstoppable
lications
Today is not a good news day for Ethereum
A vulnerability found within a popular
wallet has frozen potentially hundreds of



NUS National University of Singapore

School of Computing


Thank you!