Designing Asynchronous Multiparty Protocols with Crash-Stop Failures

Adam D. Barwell[1] Ping Hou[2] Nobuko Yoshida[2] Fangyi Zhou[2,3]

[1]University of St Andrews [2]University of Oxford [3]Imperial College London

Stardust Meeting, Glasgow
18 December 2023

University of St Andrews
UNIVERSITY OF OXFORD
Imperial College London

Concurrent and Communicating Systems

- Are **ubiquitous**
  - Mobile phones, desktops, distributed systems, the internet, &*c*.
- Can be difficult to get **right** unaided
  - Communications mismatch, deadlocks, livelocks, &*c*.

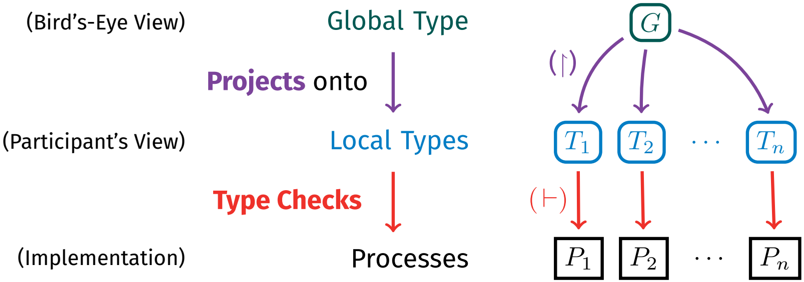
Concurrent and Communicating Systems

- Are ubiquitous
  - Mobile phones, desktops, distributed systems, the internet, &*c*.
- Can be difficult to get right **unaided**
  - Communications mismatch, deadlocks, livelocks, &*c*.
- Some assistance: **Session Types**

(Multiparty) Session Types

- Formally describe **communications behaviour** between two or more systems
- Communications behaviour is **enforced statically**

Top-Down Multiparty Session Types



Top-Down Multiparty Session Types – Advantages

» **Correct-by-construction** behavioural properties

- Communication safety
- Deadlock-freedom
- Liveness

» **Generate** protocol-conforming code

- Library support in Scala, Rust, Haskell, Erlang, OCaml, &c.

One Caveat: Fault-Tolerance, or Lack Thereof

- Traditional MPST systems assume a **perfect world**
  - No process failures
  - No message loss, duplication, or corruption

One Caveat: Fault-Tolerance, or Lack Thereof

- Traditional MPST systems assume a **perfect world**
  - No process failures
  - No message loss, duplication, or corruption
- Distributed systems can, and will, fail
  - Reality is not so convenient…

One Caveat: Fault-Tolerance, or Lack Thereof

» Traditional MPST systems assume a **perfect world**
  – No process failures
  – No message loss, duplication, or corruption

» Distributed systems can, and will, fail
  – Reality is not so convenient…

» Such MPST systems cannot reason about failures
  – How do participants handle crashes?
  – Do behavioural guarantees still hold in the presence of crashes?

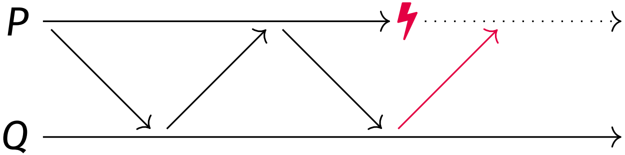
Asynchronous Multiparty Protocols with Crash-Stop Failures

We present an **asynchronous top-down** MPST theory with **crash-stop failures**.

1. Behavioural Guarantees
2. Optional Reliability Assumptions
3. Code generation toolchain, TEATRINO

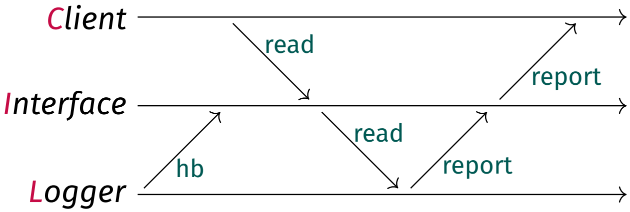
Crash-Stop Failures

- Processes can **crash arbitrarily**
- Crashed processes **make no progress** and **do not recover**
- Communications channels deliver messages in order and without losses



(Part of) A Simple Distributed Logger

A Client requests the accumulated logs from a distributed Logger via an Interface process.

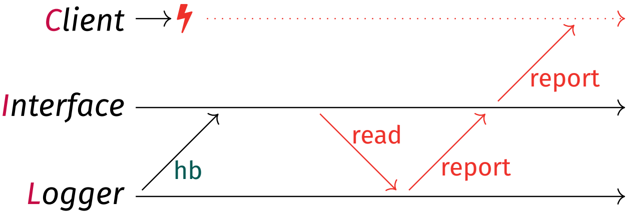


$$G = \mathtt{L}{\to}\mathtt{I}{:}\mathtt{hb}.\mathtt{C}{\to}\mathtt{I}{:}\mathtt{read}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{read}.\mathtt{L}{\to}\mathtt{I}{:}\mathtt{report}(\mathsf{log}).\mathtt{I}{\to}\mathtt{C}{:}\mathtt{report}(\mathsf{log}).\mathsf{end}$$

The full version of the protocol (and other variants) can be found in the paper/artefact.

(Part of) A Simple Distributed Logger

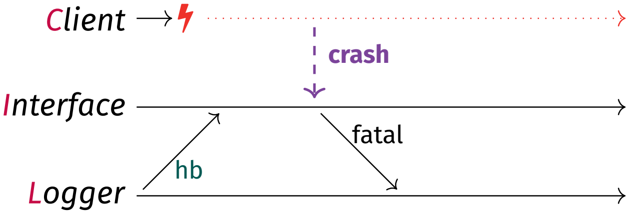
A Client requests the accumulated logs from a distributed Logger via an Interface process.



$G = \mathtt{L} \to \mathtt{I}{:}\mathtt{hb}.\ ??$

(Part of) A Simple Distributed Logger

A Client requests the accumulated logs from a distributed Logger via an Interface process.



$$G = \mathtt{L}\to\mathtt{I}:\mathtt{hb}.\mathtt{C}\to\mathtt{I}: \begin{Bmatrix} \mathtt{read} & .\mathtt{I}\to\mathtt{L}:\mathtt{read}.\mathtt{L}\to\mathtt{I}:\mathtt{report}(\mathsf{log}).\mathtt{I}\to\mathtt{C}:\mathtt{report}(\mathsf{log}).\mathsf{end} \\ \mathtt{crash} & .\mathtt{I}\to\mathtt{L}:\mathtt{fatal}.\mathsf{end} \end{Bmatrix}$$

↑ Reserved label; represents **crash detection**

Global Type Syntax

$$
\begin{array}{rcll}
B & ::= & \mathsf{int} \mid \mathsf{bool} \mid \mathsf{real} \mid \mathsf{unit} \mid \ldots & \text{Basic types} \\
G & ::= & \mathtt{p} \to \mathtt{q}^{\dagger} : \{\mathtt{m_i}(B_i).G_i\}_{i \in I} & \text{Transmission} \\
 & \mid & \mathtt{p}^{\dagger} \rightsquigarrow \mathtt{q} : j \, \{\mathtt{m_i}(B_i).G_i\}_{i \in I} \quad (j \in I) & \text{Transmission en route} \\
 & \mid & \mu \mathbf{t}.G & \text{Recursion} \\
 & \mid & \mathbf{t} & \text{Type variable} \\
 & \mid & \mathsf{end} & \text{Termination} \\
\dagger & ::= & \cdot \quad \mid \quad \lightning & \text{Crash annotation}
\end{array}
$$

» Runtime types are not available to the user, only via reduction.

» crash is a reserved label indicating crash-handling branches.

Crashing Clients

Participant C Crashes.

$$G = \mathtt{L} \to \mathtt{I}{:}\mathtt{hb}.\mathtt{C} \to \mathtt{I}{:} \begin{Bmatrix} \mathtt{read}.\mathtt{I} \to \mathtt{L}{:}\mathtt{read}.\mathtt{L} \to \mathtt{I}{:}\mathtt{report}(\mathsf{log}).\mathtt{I} \to \mathtt{C}{:}\mathtt{report}(\mathsf{log}).\mathsf{end} \\ \mathtt{crash}.\mathtt{I} \to \mathtt{L}{:}\mathtt{fatal}.\mathsf{end} \end{Bmatrix}$$

Crashing Clients

Participant C Crashes.

$$G = \mathtt{L}\to\mathtt{I}{:}\mathtt{hb}.\mathtt{C}\to\mathtt{I}{:}\begin{Bmatrix}\mathtt{read}.\mathtt{I}\to\mathtt{L}{:}\mathtt{read}.\mathtt{L}\to\mathtt{I}{:}\mathtt{report}(\mathsf{log}).\mathtt{I}\to\mathtt{C}{:}\mathtt{report}(\mathsf{log}).\mathsf{end}\\ \mathtt{crash}.\mathtt{I}\to\mathtt{L}{:}\mathtt{fatal}.\mathsf{end}\end{Bmatrix}$$

$$\downarrow_{\mathcal{R}}$$

$$G_1 = \mathtt{L}\to\mathtt{I}{:}\mathtt{hb}.\mathtt{C}^{\lightning} \rightsquigarrow \mathtt{I}{:}\begin{Bmatrix}\mathtt{read}.\mathtt{I}\to\mathtt{L}{:}\mathtt{read}.\mathtt{L}\to\mathtt{I}{:}\mathtt{report}(\mathsf{log}).\mathtt{I}\to\mathtt{C}^{\lightning}{:}\mathtt{report}(\mathsf{log}).\mathsf{end}\\ \mathtt{crash}.\mathtt{I}\to\mathtt{L}{:}\mathtt{fatal}.\mathsf{end}\end{Bmatrix}$$

Crashing Clients

Participant C Crashes.

$$G = \mathtt{L}{\to}\mathtt{I}{:}\mathtt{hb}.\mathtt{C}{\to}\mathtt{I}{:}\left\{\begin{array}{l}\mathtt{read}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{read}.\mathtt{L}{\to}\mathtt{I}{:}\mathtt{report}(\mathsf{log}).\mathtt{I}{\to}\mathtt{C}{:}\mathtt{report}(\mathsf{log}).\mathsf{end}\\ \mathtt{crash}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{fatal}.\mathsf{end}\end{array}\right\}$$

p is unreliable — p occurs in G — G is anything but a μ-term

$$\frac{\mathtt{p} \notin \mathcal{R} \qquad \mathtt{p} \in \mathsf{roles}(G) \qquad G \neq \mu\mathbf{t}.G'}{\langle \mathfrak{C}; G\rangle \xrightarrow{\mathtt{p}\lightning}_{\mathcal{R}} \langle \mathfrak{C} \cup \{\mathtt{p}\}; G\lightning\mathtt{p}\rangle}\ [\text{GR-}\lightning]$$

$\downarrow_{\mathcal{R}}$

Set of crashed roles — 'Remove' p from G

$$G_1 = \mathtt{L}{\to}\mathtt{I}{:}\mathtt{hb}.\ \mathtt{C}^{\lightning} \rightsquigarrow \mathtt{I}{:}\left\{\begin{array}{l}\mathtt{read}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{read}.\mathtt{L}{\to}\mathtt{I}{:}\mathtt{report}(\mathsf{log}).\mathtt{I}{\to}\mathtt{C}^{\lightning}{:}\mathtt{report}(\mathsf{log}).\mathsf{end}\\ \mathtt{crash}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{fatal}.\mathsf{end}\end{array}\right\}$$

But Some are More Reliable than Others

- Some participants may represent services that can be **assumed reliable**

But Some are More Reliable than Others

No processes can crash ⟷ **All** processes can crash

But what about **some** processes can crash?

- Some participants may represent services that can be **assumed reliable**

Optional Reliability Assumptions

- Specific participants can be marked as **reliable**
- Reliable participants **will not crash**
- Reduction rules, &c. conform to these assumptions

Crashing Clients

Participant I Detects

$$G_1 = \mathtt{L}{\to}\mathtt{I}{:}\mathtt{hb}.\mathtt{C}^{\lightning} \rightsquigarrow \mathtt{I}{:}\begin{Bmatrix} \mathtt{read}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{read}.\mathtt{L}{\to}\mathtt{I}{:}\mathtt{report}(\mathsf{log}).\mathtt{I}{\to}\mathtt{C}^{\lightning}{:}\mathtt{report}(\mathsf{log}).\mathsf{end} \\ \mathtt{crash}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{fatal}.\mathsf{end} \end{Bmatrix}$$

$$\downarrow^{*}_{\mathcal{R}}$$

Crashing Clients

Participant I Detects

$$G_1 = \mathtt{L}{\to}\mathtt{I}{:}\mathtt{hb}.\mathtt{C}^{\lightning} \rightsquigarrow \mathtt{I}: \begin{Bmatrix} \mathtt{read}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{read}.\mathtt{L}{\to}\mathtt{I}{:}\mathtt{report}(\mathsf{log}).\mathtt{I}{\to}\mathtt{C}^{\lightning}{:}\mathtt{report}(\mathsf{log}).\mathsf{end} \\ \mathtt{crash}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{fatal}.\mathsf{end} \end{Bmatrix}$$

$$\downarrow^{*}_{\mathcal{R}}$$

$$G_4 = \mathtt{C}^{\lightning} \rightsquigarrow \mathtt{I}: \begin{Bmatrix} \mathtt{read}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{read}.\mathtt{L}{\to}\mathtt{I}{:}\mathtt{report}(\mathsf{log}).\mathtt{I}{\to}\mathtt{C}^{\lightning}{:}\mathtt{report}(\mathsf{log}).\mathsf{end} \\ \mathtt{crash}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{fatal}.\mathsf{end} \end{Bmatrix}$$

Crashing Clients

Participant I Detects

$$G_4 = \mathtt{C}^{\lightning} \rightsquigarrow \mathtt{I}\colon \left\{ \begin{array}{l} \mathtt{read}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{read}.\mathtt{L}{\to}\mathtt{I}{:}\mathtt{report}(\mathsf{log}).\mathtt{I}{\to}\mathtt{C}^{\lightning}{:}\mathtt{report}(\mathsf{log}).\mathsf{end} \\ \mathtt{crash}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{fatal}.\mathsf{end} \end{array} \right\}$$

$$\downarrow_{\mathcal{R}}$$

$$G_5 = \mathtt{I}{\to}\mathtt{L}{:}\mathtt{fatal}.\mathsf{end}$$

Crashing Clients

Participant $\mathtt{I}$ Detects

$$G_4 = \mathtt{C}^{\lightning} \rightsquigarrow \mathtt{I}\colon \left\{ \begin{array}{l} \mathtt{read}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{read}.\mathtt{L}{\to}\mathtt{I}{:}\mathtt{report}(\mathsf{log}).\mathtt{I}{\to}\mathtt{C}^{\lightning}{:}\mathtt{report}(\mathsf{log}).\mathsf{end} \\ \mathtt{crash}.\mathtt{I}{\to}\mathtt{L}{:}\mathtt{fatal}.\mathsf{end} \end{array} \right\}$$

$$\downarrow_{\mathcal{R}} \qquad \frac{j \in I \quad \mathtt{m}_j = \mathsf{crash}}{\langle \mathfrak{C}; \mathtt{p}^{\lightning} \rightsquigarrow \mathtt{q}{:}j \left\{\mathtt{m_i}(B_i).G'_i\right\}_{i \in I} \rangle \xrightarrow{\mathtt{q} \odot \mathtt{p}}_{\mathcal{R}} \langle \mathfrak{C}; G'_j \rangle} \ [\text{GR-}\odot]$$

$$G_5 = \mathtt{I}{\to}\mathtt{L}{:}\mathtt{fatal}.\mathsf{end}$$

Local Type Syntax

B	$::=$	int $\mid$ bool $\mid$ real $\mid$ unit $\mid$ $\ldots$	Basic types
S, T	$::=$	$\mathtt{p}\&\{\mathtt{m_i}(B_i).T_i\}_{i \in I}$	External choice
	$\mid$	$\mathtt{p}\oplus\{\mathtt{m_i}(B_i).T_i\}_{i \in I}$	Internal choice
	$\mid$	$\mu\mathbf{t}.T$	Recursion
	$\mid$	$\mathbf{t}$	Type variable
	$\mid$	end	Termination
	$\mid$	stop	Crash

- The runtime stop type is the type of crashed participants
- crash is a reserved label indicating crash-handling branches

Type Checking Distributed Logging with Local Types

$$T_{\mathtt{C}} = \mathtt{I}\oplus\mathtt{read}.\mathtt{I}\&\mathtt{report}(\mathsf{log}).\mathsf{end}$$

$$P_{\mathtt{C}} = \mathtt{I}!\mathtt{read}.\mathtt{I}?\mathtt{report}(x).\mathbf{0}$$

$$T_{\mathtt{I}} = \mathtt{L}\&\mathtt{hb}.\mathtt{C}\&\begin{Bmatrix}\mathtt{read}.\mathtt{L}\oplus\mathtt{read}.\mathtt{L}\&\mathtt{report}(\mathsf{log}).\mathtt{C}\oplus\mathtt{report}(\mathsf{log}).\mathsf{end}\\ \mathsf{crash}.\mathtt{L}\oplus\mathtt{fatal}.\mathsf{end}\end{Bmatrix}$$

$$P_{\mathtt{I}} = \mathtt{L}?\mathtt{hb}.\sum\begin{Bmatrix}\mathtt{C}?\mathtt{read}.\mathtt{L}!\mathtt{read}.\mathtt{L}?\mathtt{report}(x).\mathtt{C}!\mathtt{report}\langle x\rangle.\mathbf{0}\\ \mathtt{C}?\mathsf{crash}.\mathtt{L}!\mathtt{fatal}.\mathbf{0}\end{Bmatrix}$$

We largely elide the session π-calculus here, but full details can be found in the paper.

Local Type Context Reduction

$$\begin{aligned} T_{\mathtt{C}} &= \mathtt{I}\oplus\mathtt{read}.\mathtt{I}\&\mathtt{report}(\mathsf{log}).\mathsf{end} \\ P_{\mathtt{C}} &= \mathtt{I}!\mathtt{read}.\mathtt{I}?\mathtt{report}(x).\mathbf{0} \end{aligned} \quad \rightarrow \quad \begin{aligned} T_{\mathtt{C}} &= \mathsf{stop} \\ P_{\mathtt{C}} &= \lightning \end{aligned}$$

- Rules are largely standard
  - Crashing, crash-detection, &c. rules are novel
- Queues handle asynchronous message passing
  - Queues are made unavailable when the corresponding participant has crashed

Projecting Global Types into Local Types

Relates G and T for a given $\mathtt{p}$.

$$(\mathtt{q}{\to}\mathtt{r}^{\dagger}\colon \{\mathtt{m_i}(B_i).G_i\}_{i\in I}) \upharpoonright_{\mathcal{R}} \mathtt{p} = \begin{cases} \mathtt{r}\oplus\{\mathtt{m_i}(B_i).(G_i \upharpoonright_{\mathcal{R}} \mathtt{p})\}_{i\in\{j\in I \,|\, \mathtt{m}_j\neq \mathtt{crash}\}} & \text{if } \mathtt{p}=\mathtt{q} \\ \mathtt{q}\&\{\mathtt{m_i}(B_i).(G_i \upharpoonright_{\mathcal{R}} \mathtt{p})\}_{i\in I} & \text{if } \mathtt{p}=\mathtt{r}, \text{ and } \mathtt{q}\notin\mathcal{R} \text{ implies } \exists k\in I : \mathtt{m}_k = \mathtt{crash} \\ \bigsqcap_{i\in I} G_i \upharpoonright_{\mathcal{R}} \mathtt{p} & \text{if } \mathtt{p}\neq\mathtt{q}, \text{ and } \mathtt{p}\neq\mathtt{r} \end{cases}$$

$$(\mathtt{q}^{\dagger}{\rightsquigarrow}\mathtt{r}{:}j\, \{\mathtt{m_i}(B_i).G_i\}_{i\in I}) \upharpoonright_{\mathcal{R}} \mathtt{p} = \begin{cases} G_j \upharpoonright_{\mathcal{R}} \mathtt{p} & \text{if } \mathtt{p}=\mathtt{q} \\ \mathtt{q}\&\{\mathtt{m_i}(B_i).(G_i \upharpoonright_{\mathcal{R}} \mathtt{p})\}_{i\in I} & \text{if } \mathtt{p}=\mathtt{r}, \text{ and } \mathtt{q}\notin\mathcal{R} \text{ implies } \exists k\in I : \mathtt{m}_k = \mathtt{crash} \\ \bigsqcap_{i\in I} G_i \upharpoonright_{\mathcal{R}} \mathtt{p} & \text{if } \mathtt{p}\neq\mathtt{q}, \text{ and } \mathtt{p}\neq\mathtt{r} \end{cases}$$

$$(\mu\mathbf{t}.G) \upharpoonright_{\mathcal{R}} \mathtt{p} = \begin{cases} \mu\mathbf{t}.(G \upharpoonright_{\mathcal{R}} \mathtt{p}) & \text{if } \mathtt{p}\in G \text{ or } \mathrm{fv}(\mu\mathbf{t}.G)\neq\emptyset \\ \mathtt{end} & \text{otherwise} \end{cases}$$

$$\mathbf{t} \upharpoonright_{\mathcal{R}} \mathtt{p} = \mathbf{t}$$

$$\mathtt{end} \upharpoonright_{\mathcal{R}} \mathtt{p} = \mathtt{end}$$

The (full) merging operator definition can be found in the paper.

Projection Begat Properties

1. Association of Global Type and Configuration (Soundness and Completeness)
   - Global and local types do not diverge when reducing

Formal definitions and proofs can be found in the paper.

Projection Begat Properties

1. Association of Global Type and Configuration (Soundness and Completeness)
   - Global and local types do not diverge when reducing

2. Configuration Safety
   - There are no label mismatches
   - Each receiver must be able to handle the potential crash of the (unreliable) sender

Formal definitions and proofs can be found in the paper.

Projection Begat Properties

1. Association of Global Type and Configuration (Soundness and Completeness)
   - Global and local types do not diverge when reducing
2. Configuration Safety
   - There are no label mismatches
   - Each receiver must be able to handle the potential crash of the (unreliable) sender
3. Deadlock-Freedom (Progress)
   - Local types are able to reduce until they terminate or crash

Formal definitions and proofs can be found in the paper.

Projection Begat Properties

1. Association of Global Type and Configuration (Soundness and Completeness)
   - Global and local types do not diverge when reducing
2. Configuration Safety
   - There are no label mismatches
   - Each receiver must be able to handle the potential crash of the (unreliable) sender
3. Deadlock-Freedom (Progress)
   - Local types are able to reduce until they terminate or crash
4. Liveness
   - Every pending internal/external choice is eventually triggered (by message transmission or crash detection)

Formal definitions and proofs can be found in the paper.

Asynchronous Multiparty Protocols with Crash-Stop Failures

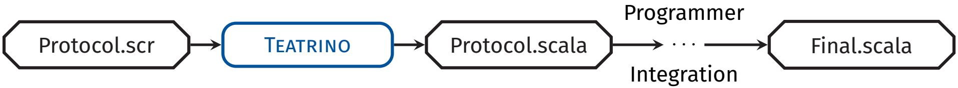
We present an **asynchronous top-down** MPST theory with **crash-stop failures**.

✓ Behavioural Guarantees

✓ Optional Reliability Assumptions

⇒ Code generation toolchain, Teatrino

Teatrino



- Scribble-inspired **code generation toolchain**
  - Consumes Scribble protocols
  - Produces protocol-conforming Scala code using the Effpi concurrency library
  - Generated code is executable
- **Implements** our (non-runtime) Global and Local Types, and Projection
- **Extends** both the Effpi and Scribble syntaxes with crash-stop failures

The (Extended) Scribble Protocol Description Language

- We support (a less sugary subset of) the version accepted by νScr[1]
  - No support for `do`-notation – recursion is expressed via `rec` and `continue`
  - No support for auxiliary protocol definitions

Extensions

1. `reliable` role declarations
2. Reserved `crash` label

[1] https://nuscr.dev/

The (Extended) Scribble Protocol Description Language

$$G = \mathtt{L}{\rightarrow}\mathtt{I}{:}\mathtt{hb}.\mathtt{C}{\rightarrow}\mathtt{I}{:}\left\{\begin{array}{l}\mathtt{read}.\mathtt{I}{\rightarrow}\mathtt{L}{:}\mathtt{read}.\mathtt{L}{\rightarrow}\mathtt{I}{:}\mathtt{report}(\mathsf{log}).\mathtt{I}{\rightarrow}\mathtt{C}{:}\mathtt{report}(\mathsf{log}).\mathsf{end} \\ \mathtt{crash}.\mathtt{I}{\rightarrow}\mathtt{L}{:}\mathtt{fatal}.\mathsf{end}\end{array}\right\}$$

```
global protocol G(reliable role L, role C, reliable role I) {
  hb from L to I;
  choice at C {
    read from C to I;
    read from I to L;
    report(Log) from L to I;
    report(Log) from I to C;
  } or {
    crash from C to I;
    fatal from I to L;
  }
}
```

The (Extended) EFFPI Concurrency Library

- Embedded Domain Specific Language for SCALA 3
- Leverages type features to **represent local types directly in code**
  - Union types, match types, and dependent and polymorphic function types

Extensions

1. Support for crash-handling branches
   - New type-level receive construct: `InErr`
   - New value-level receive construct: `receiveErr`
2. Support for crash detection
   - Implemented using timeouts

Original version of EFFPI: `https://github.com/alcestes/effpi`

The (Extended) Effpi Concurrency Library

$$T_{\mathtt{I}} = \mathtt{L}\&\mathtt{hb}.\mathtt{C}\&\begin{Bmatrix}\mathtt{read}.\mathtt{L}\oplus\mathtt{read}.\mathtt{L}\&\mathtt{report}(\mathsf{log}).\mathtt{C}\oplus\mathtt{report}(\mathsf{log}).\mathsf{end}\\ \mathsf{crash}.\mathtt{L}\oplus\mathtt{fatal}.\mathsf{end}\end{Bmatrix}$$

```
type I[C0 <: InChan[Hb],
       C1 <: OutChan[Fatal],
       C2 <: InChan[Read],
       C3 <: InChan[Report],
       C4 <: OutChan[Report]] =
  In[C0, Hb, (X <: Hb) =>
    InErr[C2, Read,
      (Y <: Read) =>
        Out[C3,Read] >>: In[C4, Report, (Z <: Log) => Out[C5, Report]],
      (Err <: Throwable) => Out[C2,Fatal]
  ]]
```

Evaluating TEATRINO

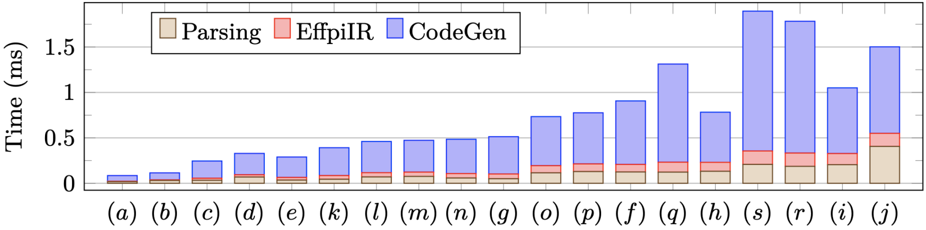
Expressiveness

- Applied to examples from session type and distributed systems literature
- Standard examples extended with crash-handling behaviour
- Two patterns: **graceful failure** and **failover**

Feasibility

- We give generation times for all of our examples
- We report times for the three main generation phases:
  1. Parsing
  2. EffpiIR Generation
  3. Code Generation

Evaluating TEATRINO



» Evaluated on 19 protocols taken from session type and distributed system literature
  – Code generation times all under 3 milliseconds

Summary

- Asynchronous top-down MPST theory with crash-stop failures:
  - Support for fully-reliable to fully-unreliable protocols
  - Safety, deadlock-freedom, and liveness guarantees

- TEATRINO: toolchain support for generating protocol-conformant SCALA code

- Future work:
  - Investigate different crash and failure models (e.g. crash-recover, link failures)

Links

- Full version: `https://arxiv.org/abs/2305.06238`
- Artefact: `https://doi.org/10.5281/zenodo.7714132`
- Artefact Source: `https://github.com/adbarwell/ECOOP23-Artefact`