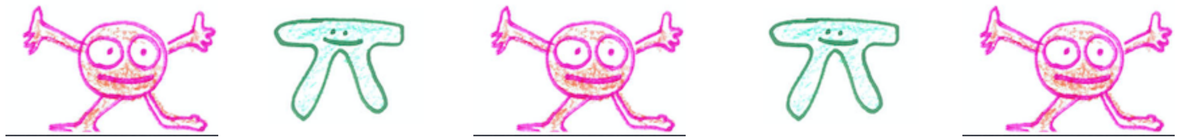
Deadlock-Free Asynchronous Message Reordering in Rust with Multiparty Session Types

27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming **[PPoPP 2022]**



Zak Cutner, NY and Martin Vassor

Kent University, 30th January 2023

Communications are Ubiquitous

- Increasingly, **communications** are the way to organise software and systems.
- Industry trend – programming languages with **explicit message-passing primitives.**

microservices

Problems: Concurrency Bugs

- Communications increase **concurrency bugs**
  - Survey of 4k users [golang.org]
  - Analysis of 6 large software systems [ASPLOS 19] [PLDI 22]

Uber

The Go Gopher

Do not communicate by sharing memory; share memory by communicating

– Go Philosophy

Problems: Concurrency Bugs

- Communications increase **concurrency bugs**
  - Survey of 4K users **[golang.org]**
  - Analysis of 6 large software systems **[ASPLOS 19]**

Uber's 14 million lines of Go hosting 2100 microservices [PLDI 22]

More than a half of concurrency bugs in Go are caused by communications.

deadlock

channel errors

The Go Gopher

Problems: Concurrency Bugs

- Communications increase **concurrency bugs**
  - Survey of 4k users [golang.org]
  - Analysis of 6 large software systems [ASPLOS 19] [PLDI 22]

More than a half of concurrency bugs in Go are caused by communications.

Session Types

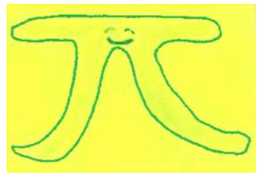
- Prevent concurrency bugs.
- Can abstract, implement and manage communications as **Protocols.**
- **Clean, Cheap** and **Retrofittable.**

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Milner, Honda, NY

Binary Session Types — ESOP'98



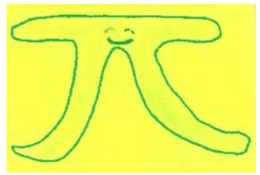
↓

Joined W3C Standardization — 2002

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



↓

Joined W3C Standardization — 2002

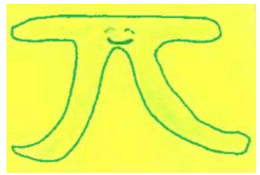
↓

Multiparty Session Types — POPL' 08

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



↓

Joined W3C Standardization — 2002

↓

Multiparty Session Types — POPL' 08

↓

largest open source company in the world

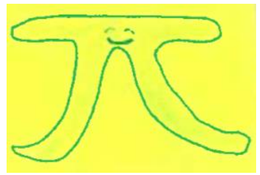
Red Hat

Scribble

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



Joined W3C Standardization — 2002

Multiparty Session Types — POPL' 08

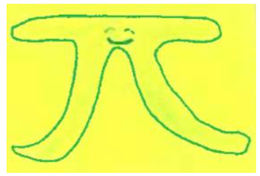
25M US$ large cyber infrastructures

middleware standardisations

Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

Binary Session Types — ESOP'98



Joined W3C Standardization — 2002

Multiparty Session Types — POPL' 08







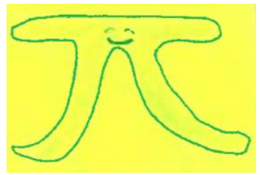



Why Session Types, Why Now?

Significant academic and industry interests via fundamental breakthroughs

ETAPS Test Time Award 2019

Binary Session Types — ESOP'98



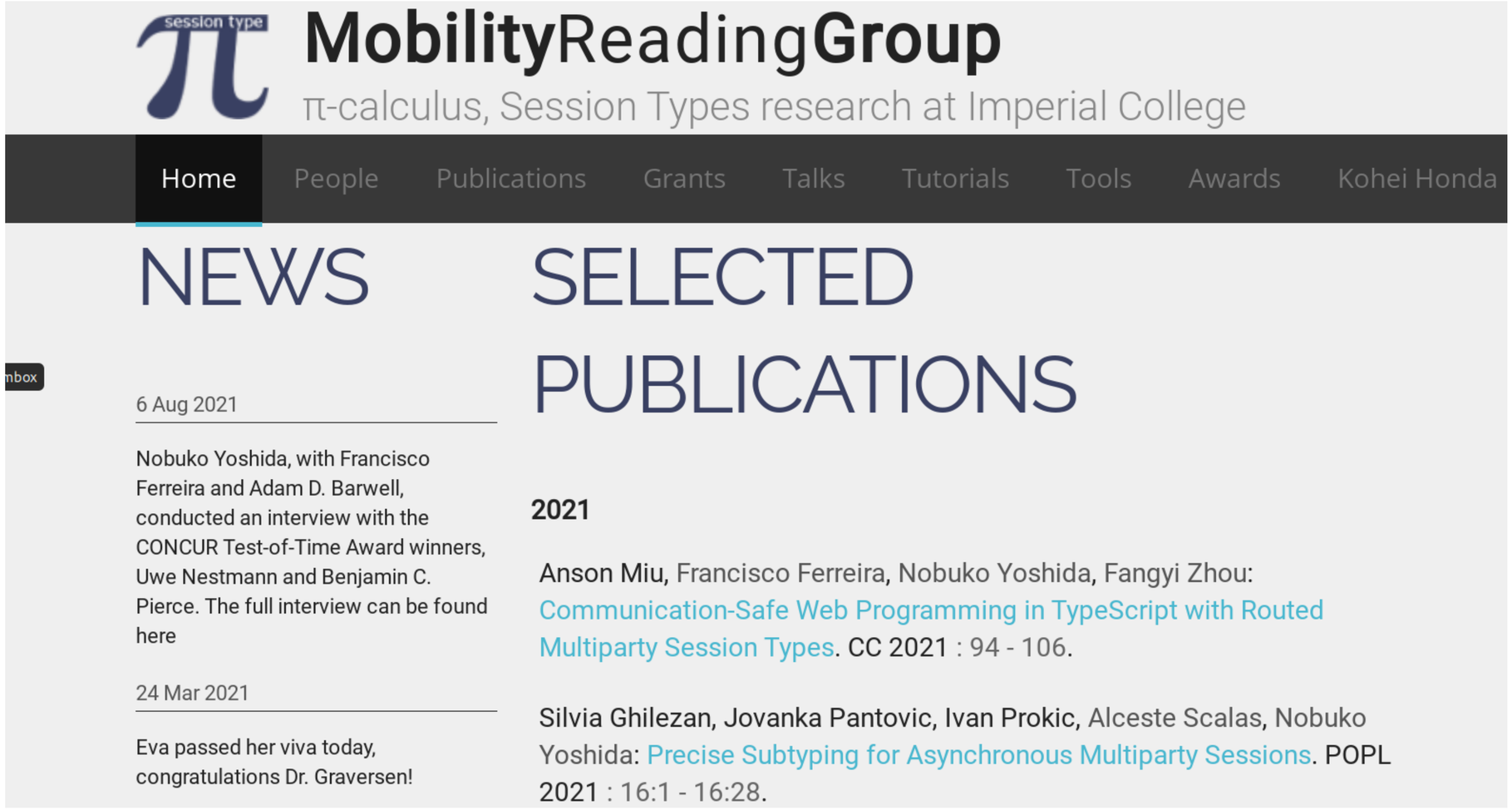
Joined W3C Standardization — 2002

POPL Influential Paper Award 2018

Multiparty Session Types — POPL' 08

Mobility Reading Group

http://mrg.cs.ox.ac.uk/



Introduction

Rust Language

- Modern systems language focussed on safety and performance
- “Most loved language” for past five years on StackOverflow
- Particular emphasis on safe concurrency using message passing
- Affine type system is well-suited to session types

Ring Protocol

Example

Global Type

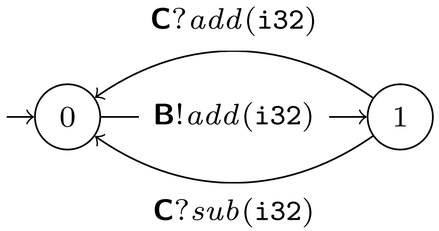
$$\mathsf{G} = \mu \mathtt{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\mathtt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\mathtt{i32}).\mathbf{C} \to \mathbf{A} : \{ add(\mathtt{i32}).\mathtt{t} \} \\ sub(\mathtt{i32}).\mathbf{C} \to \mathbf{A} : \{ sub(\mathtt{i32}).\mathtt{t} \} \end{array} \right\} \right\}$$

Ring Protocol

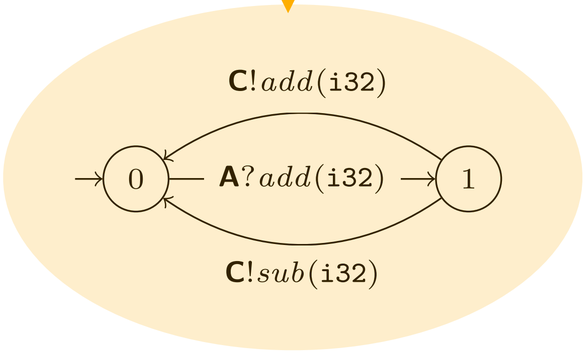
Example

$$\mathsf{G} = \mu \mathtt{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\mathtt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\mathtt{i32}).\mathbf{C} \to \mathbf{A} : \{ add(\mathtt{i32}).\mathtt{t} \} \\ sub(\mathtt{i32}).\mathbf{C} \to \mathbf{A} : \{ sub(\mathtt{i32}).\mathtt{t} \} \end{array} \right\} \right\}$$

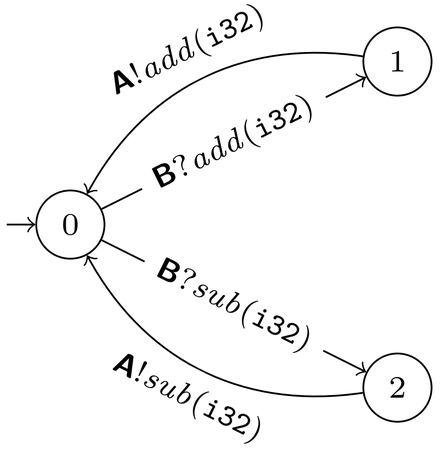
PROJECTION



PROJECTION



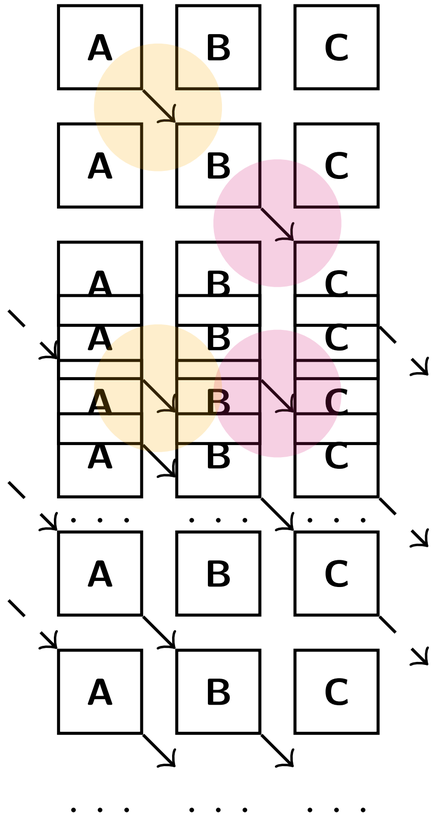
PROJECTION



Challenge

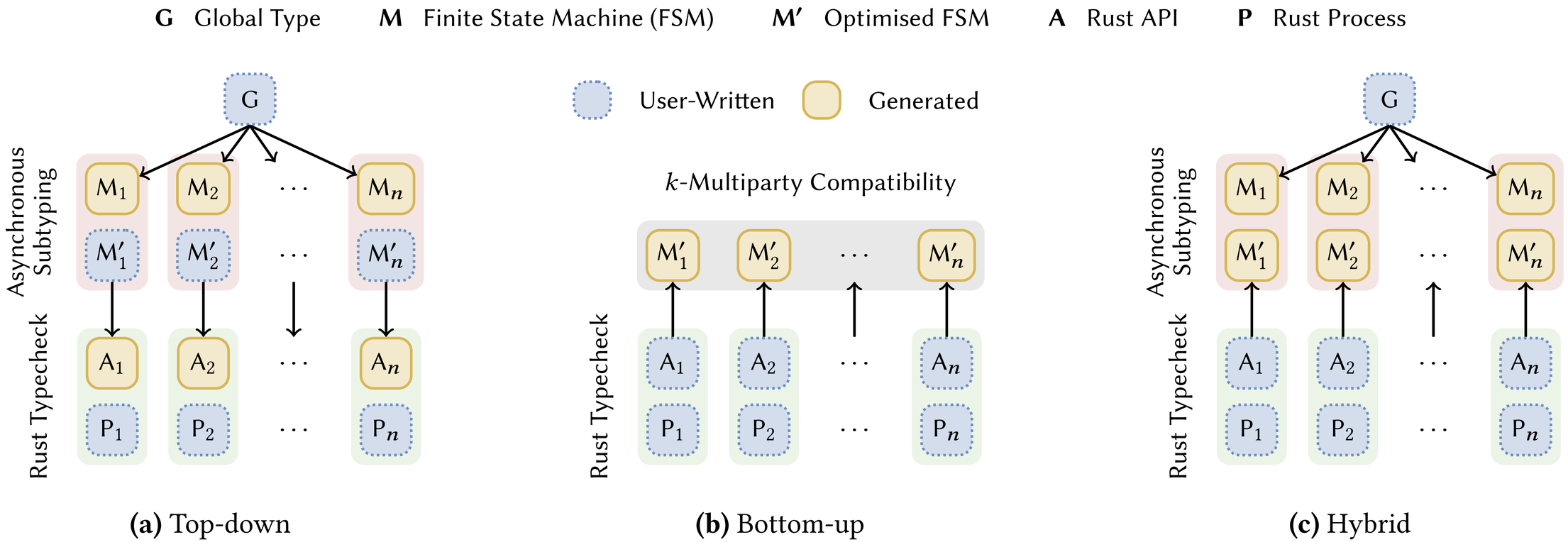
Asynchronous Orderings

- Global types are inherently synchronous
  - Projection provides only one possible ordering
- Interactions can be reordered for efficiency while preserving safety
  1. Data dependencies must be preserved
  2. Sound and practical asynchronous reordering rules must be found



Rumpsteak Framework

Three Approaches



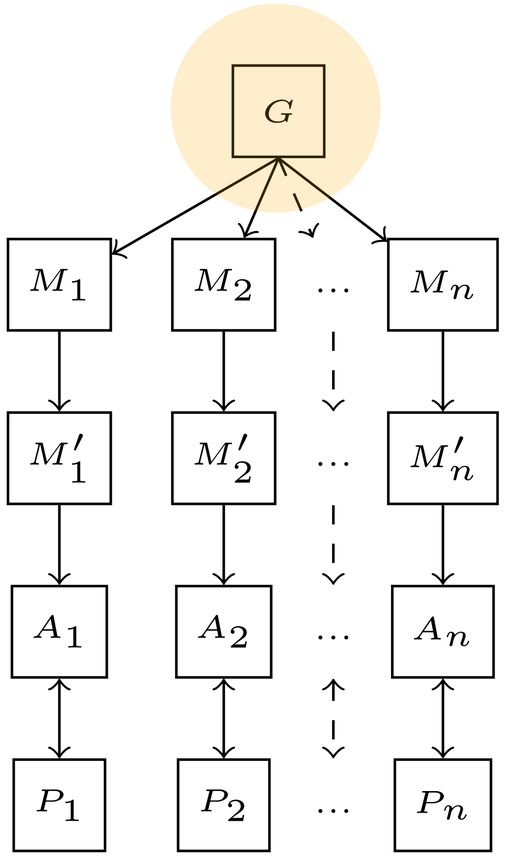
(a) Top-down

(b) Bottom-up

(c) Hybrid

Workflow

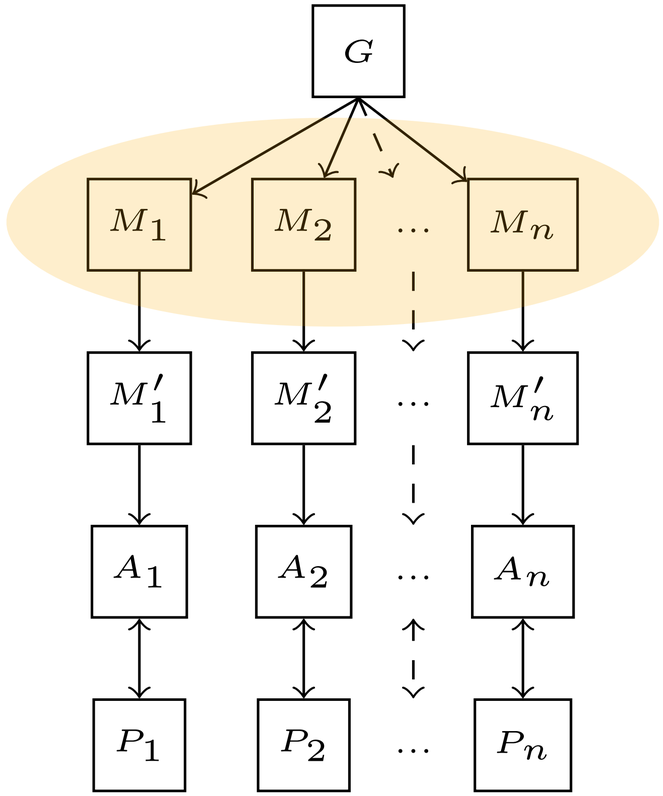
Top-Down Approach



Global Type

Workflow

Top-Down Approach

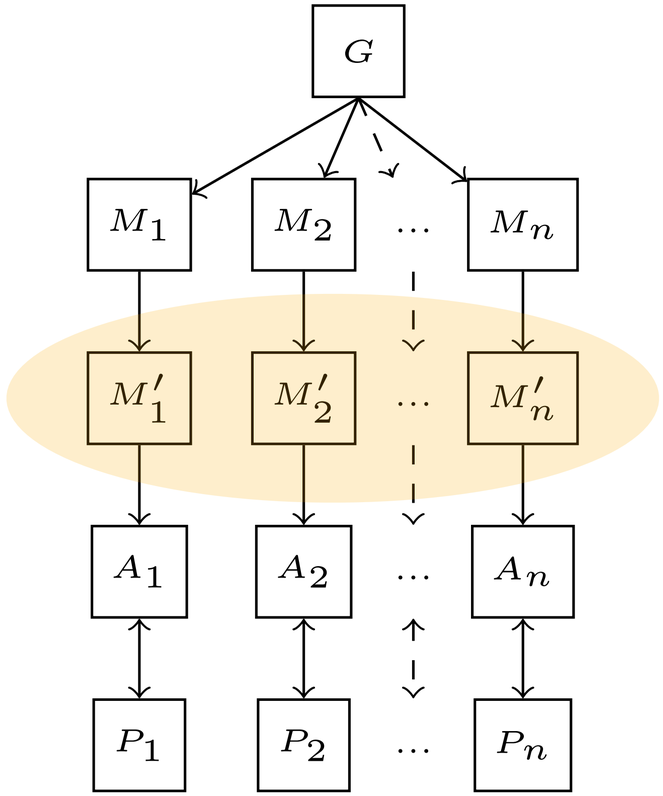


Global Type

Projected FSM

Workflow

Top-Down Approach



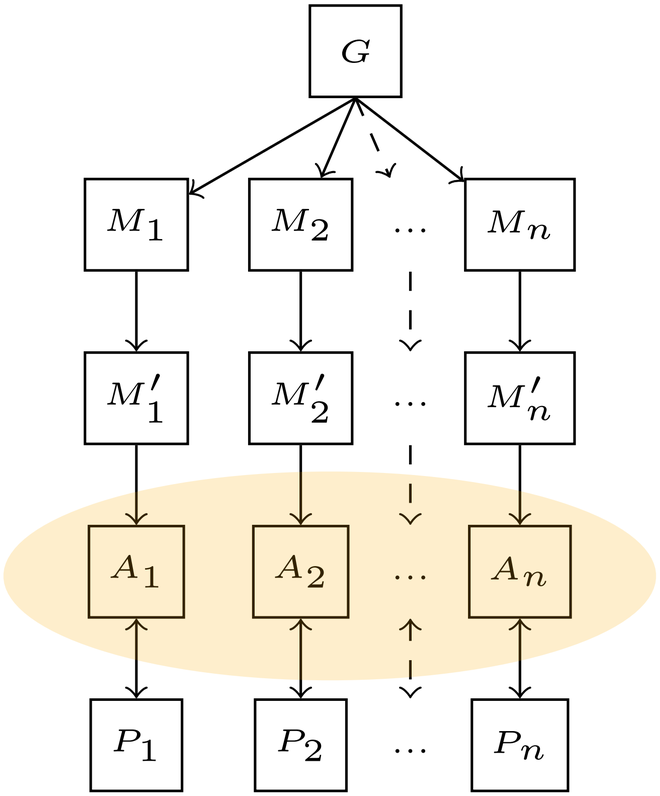
Global Type

Projected FSM

Optimised FSM

Workflow

Top-Down Approach



Global Type

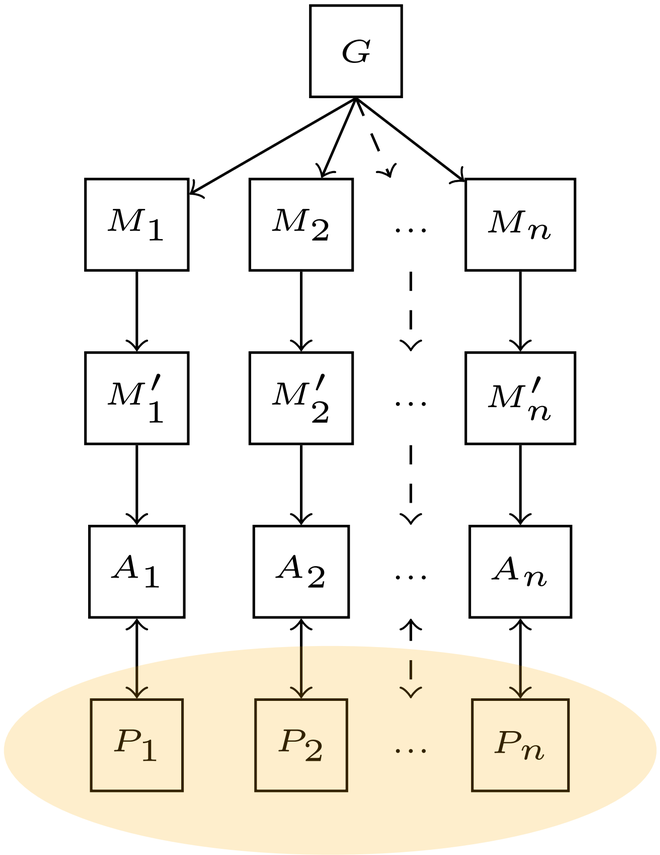
Projected FSM

Optimised FSM

Rust API

Workflow

Top-Down Approach



Global Type

Projected FSM

Optimised FSM

Rust API

Rust Implementation

Ring Protocol

Example

Global Type

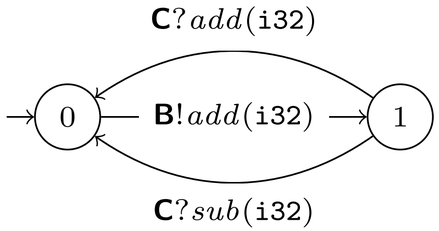
$$\mathsf{G} = \mu \mathbf{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\texttt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{ add(\texttt{i32}).\mathbf{t} \} \\ sub(\texttt{i32}).\mathbf{C} \to \mathbf{A} : \{ sub(\texttt{i32}).\mathbf{t} \} \end{array} \right\} \right\}$$

Ring Protocol

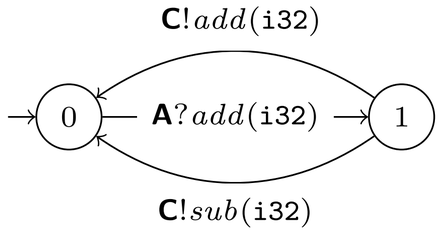
Example

$$\mathsf{G} = \mu \mathtt{t}.\mathbf{A} \to \mathbf{B} : \left\{ add(\mathtt{i32}).\mathbf{B} \to \mathbf{C} : \left\{ \begin{array}{l} add(\mathtt{i32}).\mathbf{C} \to \mathbf{A} : \{ add(\mathtt{i32}).\mathtt{t} \} \\ sub(\mathtt{i32}).\mathbf{C} \to \mathbf{A} : \{ sub(\mathtt{i32}).\mathtt{t} \} \end{array} \right\} \right\}$$

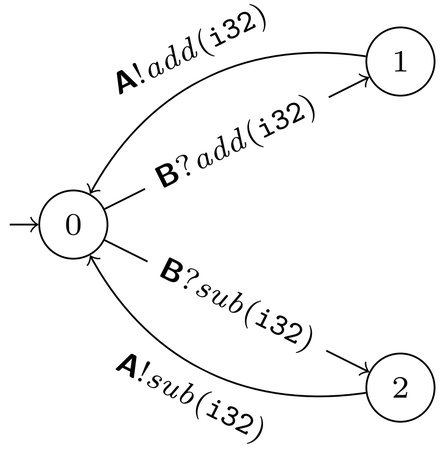
PROJECTION



PROJECTION

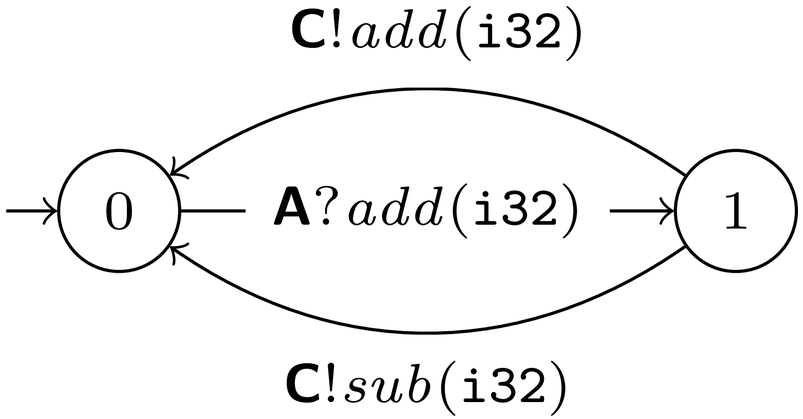


PROJECTION



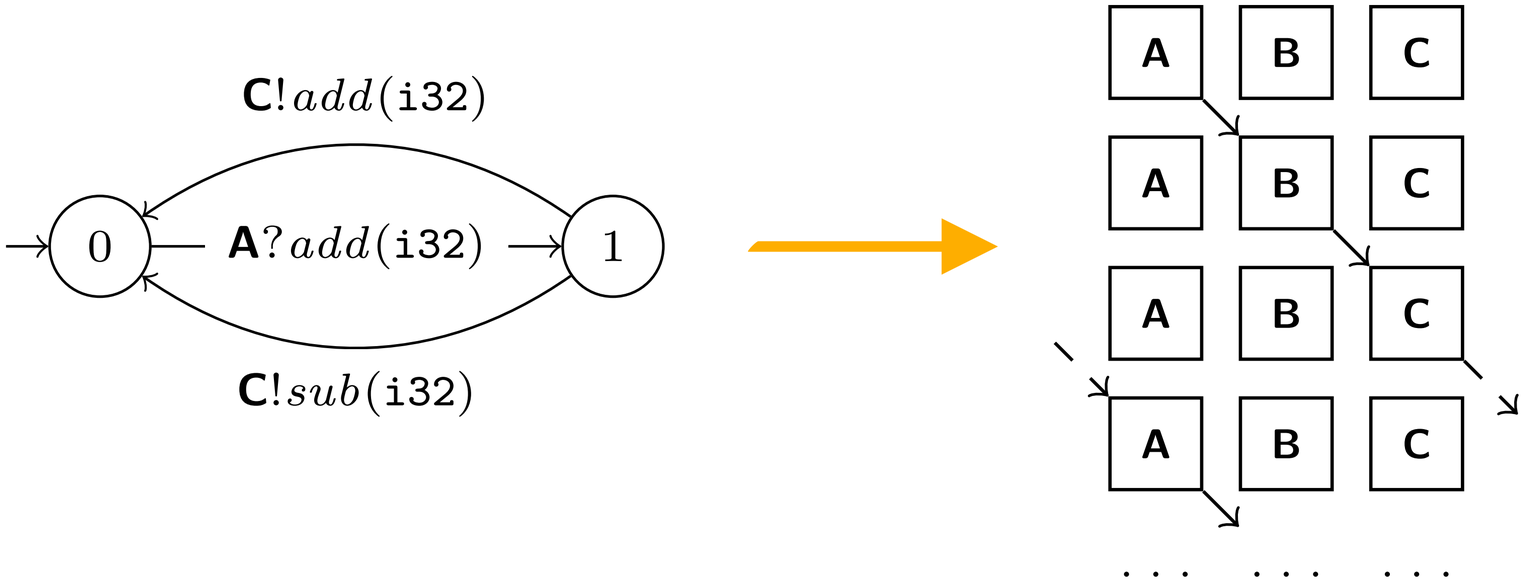
Ring Protocol

Example



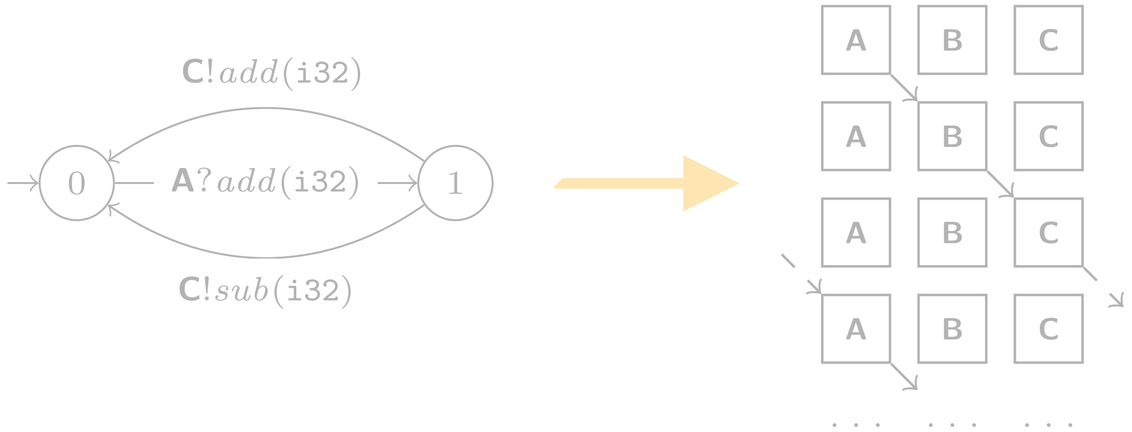
Ring Protocol

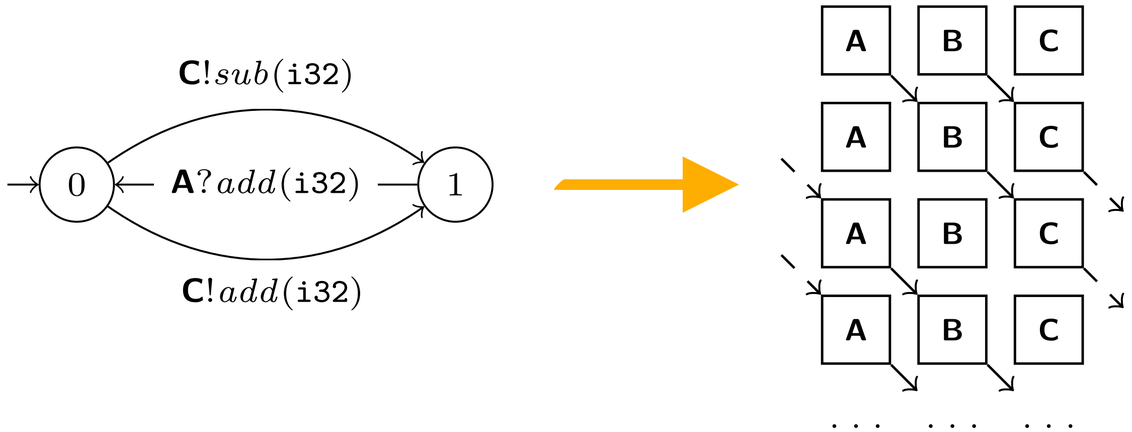
Example



Ring Protocol

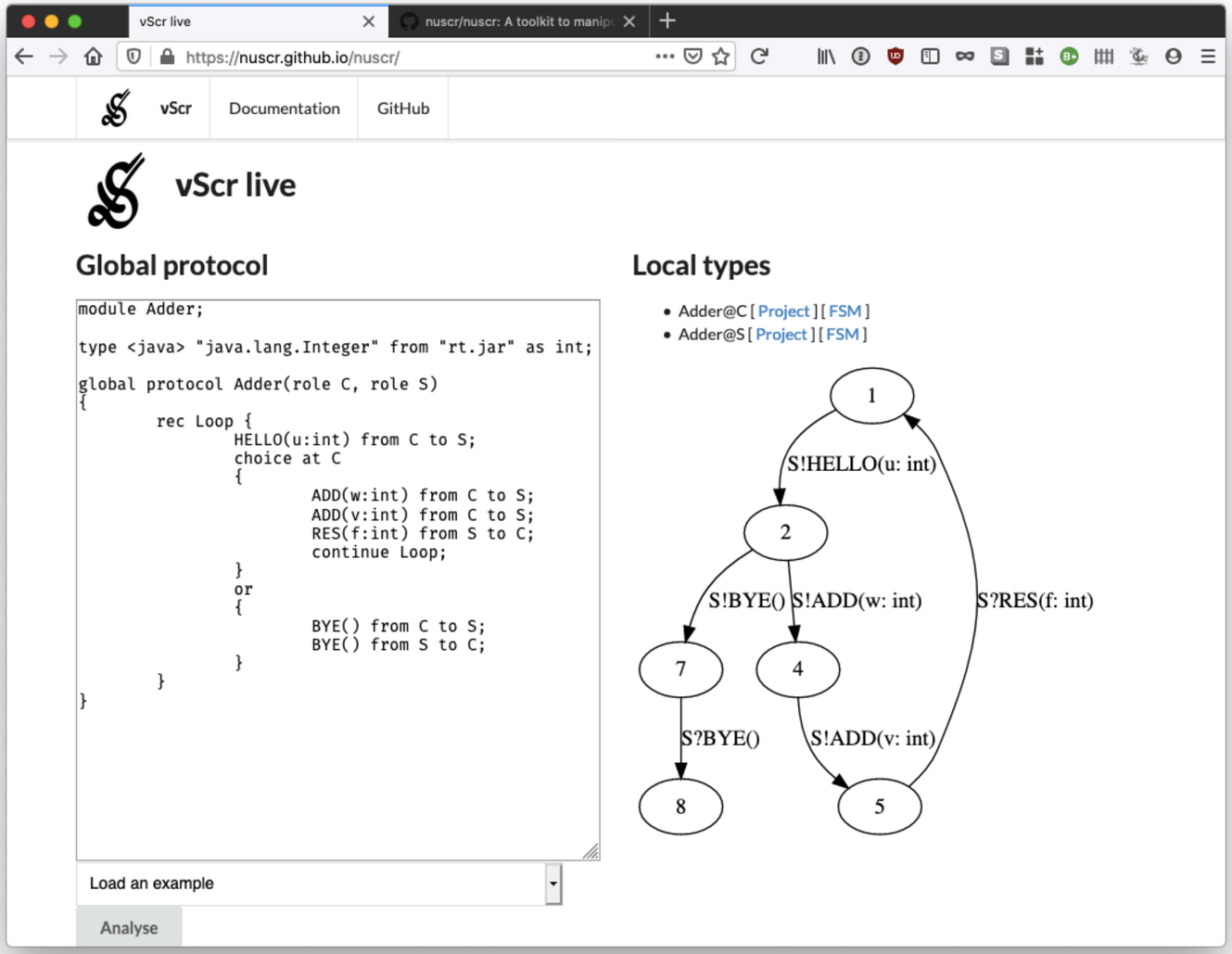
Example





vScr An Extensible Toolchain for Multiparty Session Types

- It’s small and easy to modify
- Available on opam
  - opam install nuscr
- Available on GitHub
  - https://github.com/nuscr
- Available on the web
  - https://nuscr.dev



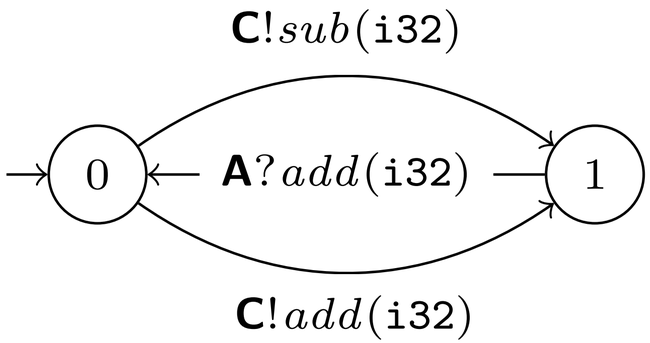
Scribble

Protocol Description Language

```
global protocol Ring(role A, role B, role C) {
    Add(i32) from A to B;
    choice at B {
        Add(i32) from B to C;
        Add(i32) from C to A;
        do Ring(A, B, C);
    } or {
        Sub(i32) from B to C;
        Sub(i32) from C to A;
        do Ring(A, B, C);
    }
}
```

Ring Protocol

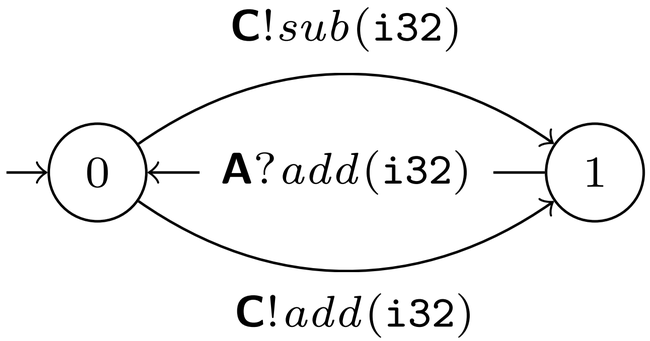
Rust API




```rust
#[derive(Role)]
#[message(Label)]
struct B(#[route(A)] Receiver, #[route(C)] Sender);
```

Ring Protocol

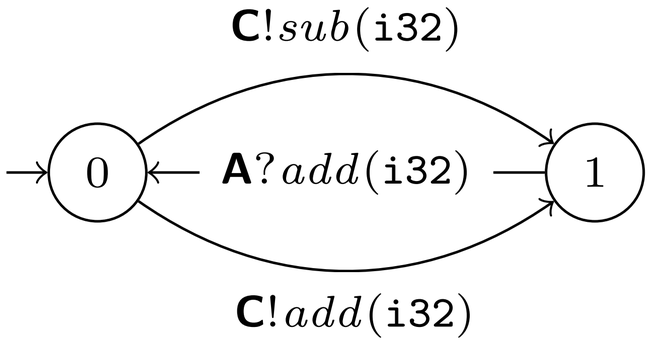
Rust API




```rust
#[derive(Role)]
#[message(Label)]
struct B(#[route(A)] Receiver, #[route(C)] Sender);
```

Ring Protocol

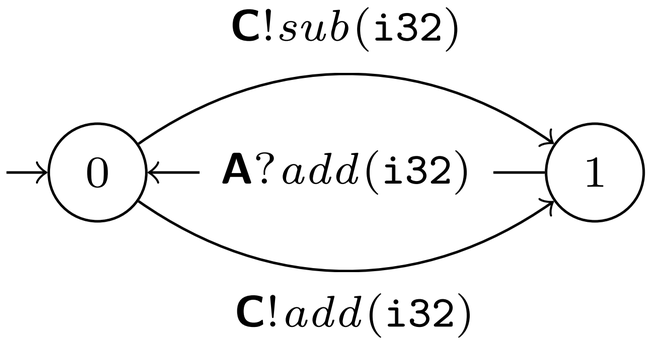
Rust API




```rust
#[derive(Role)]
#[message(Label)]
struct B(#[route(A)] Receiver, #[route(C)] Sender);

#[derive(Message)]
enum Label {
    Add(Add),
    Sub(Sub),
}

struct Add(i32);
struct Sub(i32);
```

Ring Protocol

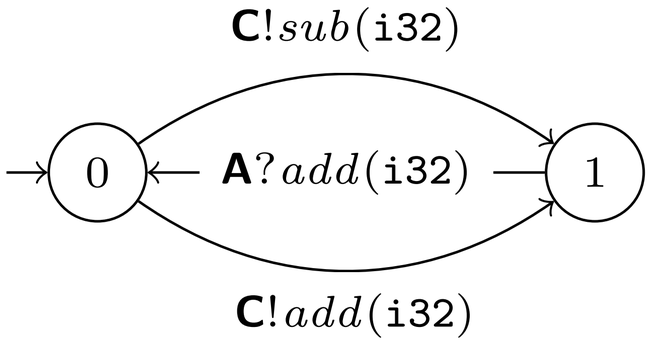
Rust API




```rust
#[derive(Role)]
#[message(Label)]
struct B(#[route(A)] Receiver, #[route(C)] Sender);

#[derive(Message)]
enum Label {
    Add(Add),
    Sub(Sub),
}

struct Add(i32);
struct Sub(i32);

#[session]
type RingB = Select<C, RingBChoice>;

#[session]
enum RingBChoice {
    Add(Add, Receive<A, Add, RingB>),
    Sub(Sub, Receive<A, Add, RingB>),
}
```

Ring Protocol

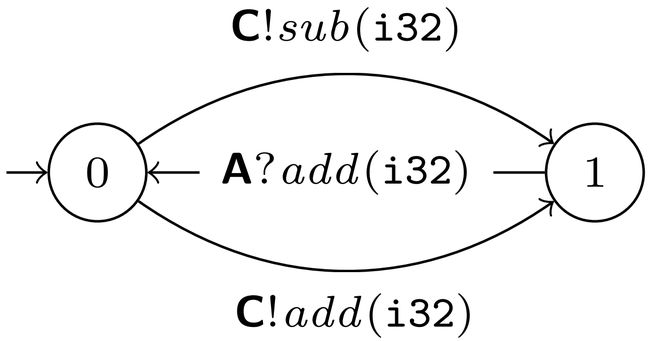
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

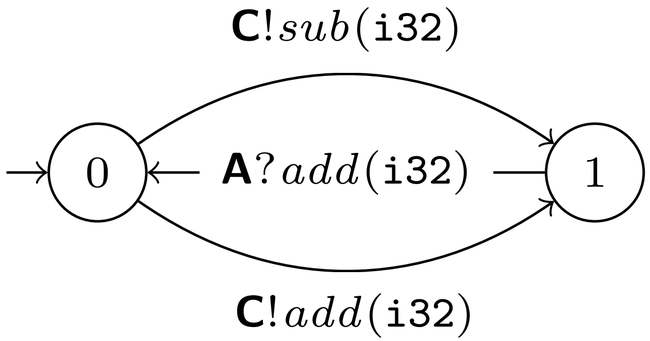
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

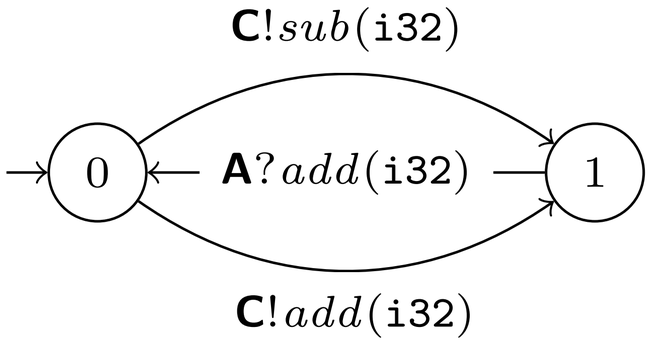
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

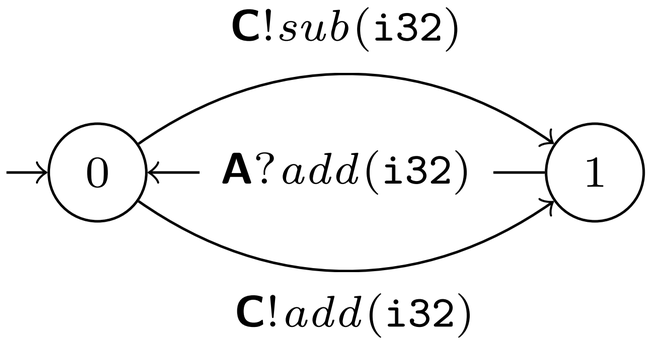
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

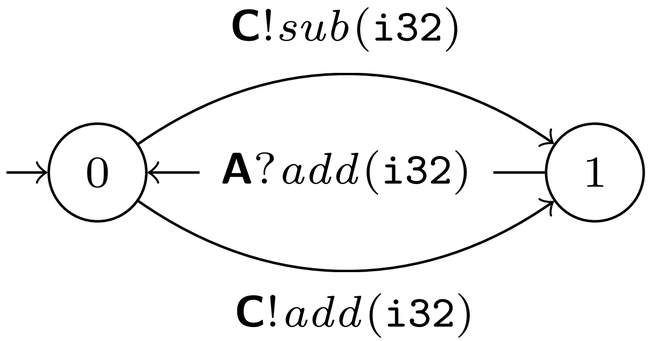
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

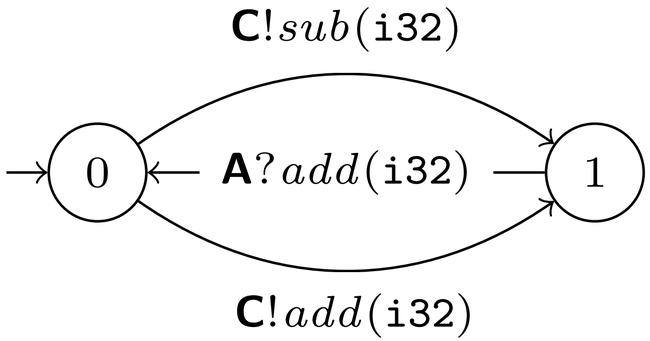
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

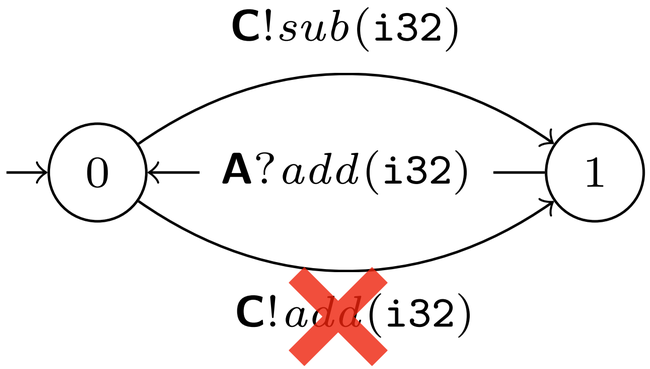
Implementation




```rust
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

Ring Protocol

Implementation

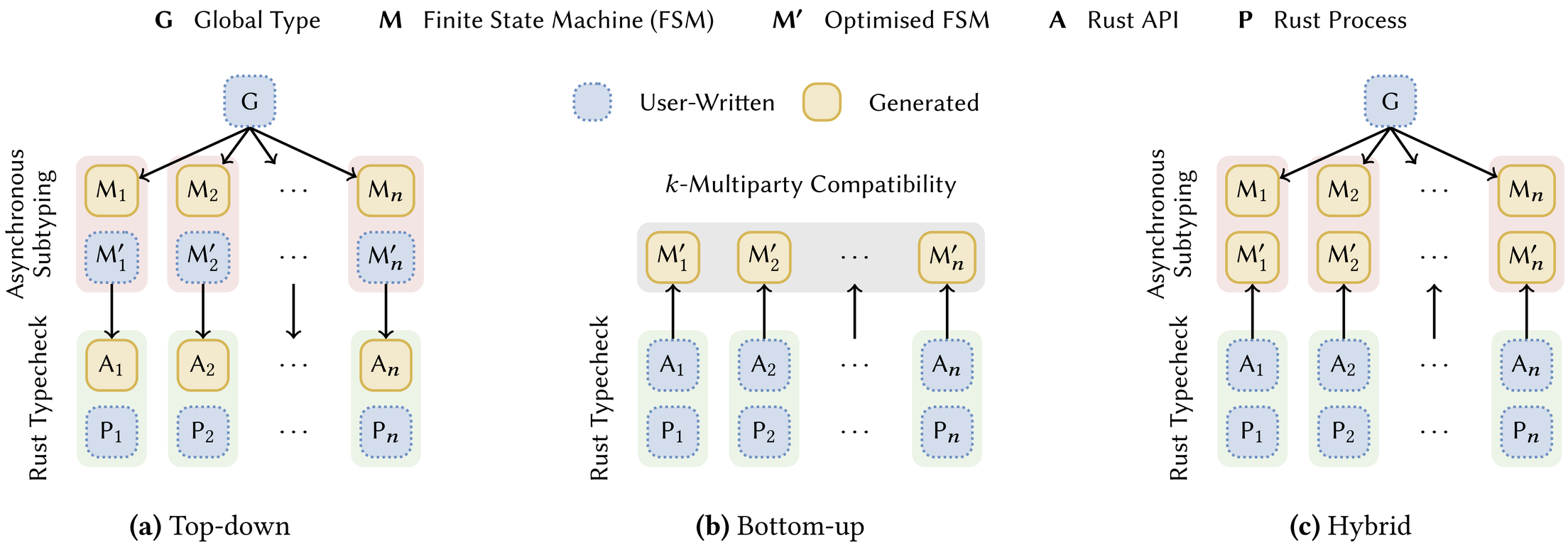



```
async fn ring_b(
    role: &mut B,
    mut input: i32,
) -> Result<Infallible> {
    try_session(role, |mut s: RingB<'_, _>| async {
        loop {
            let x = input * 2;

            s = if x > 0 {
                let s = s.select(Add(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y + x;
                s
            } else {
                let s = s.select(Sub(x)).await?;
                let (Add(y), s) = s.receive().await?;
                input = y - x;
                s
            };
        }
    })
    .await
}
```

```
method not found in `rumpsteak::Select<'_, B, C, RingBChoice<'_, B>>`
```

Rumpsteak Framework

Three Approaches



(a) Top-down

(b) Bottom-up

(c) Hybrid

Theories for Communication Optimisation

Asynchronous Reordering Revisited

How do we check that asynchronous reorderings are safe?

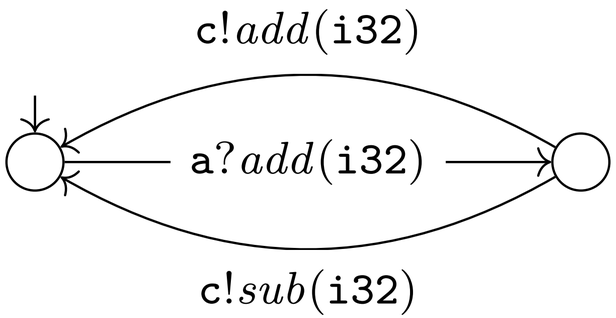
1. Asynchronous subtyping [Ghilezan, Pantvic, Prokic, Scalas and NY **POPL'2021**]

2. *k*-multiparty compatibility [Lange and NY, **CAV'2019**]

Safety

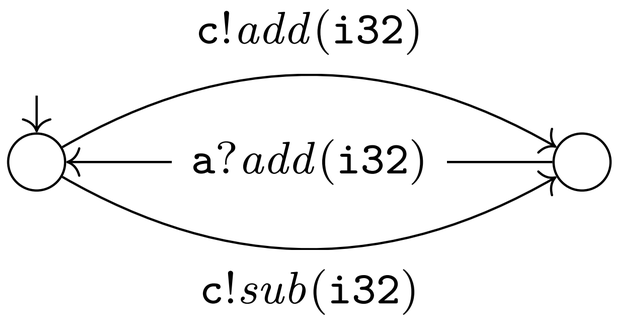
Asynchronous Subtyping

PROJECTED B



Safe?

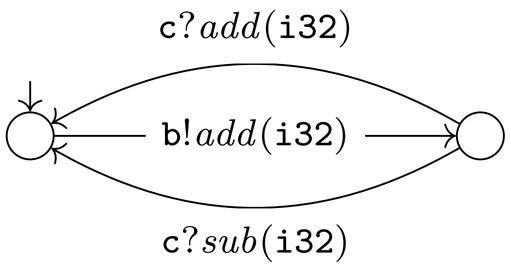
OPTIMISED B



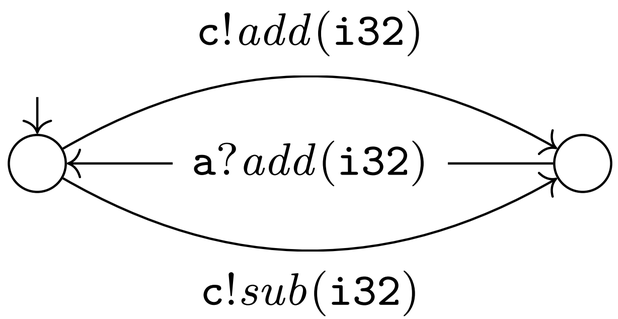
Safety

k-Multiparty Compatibility

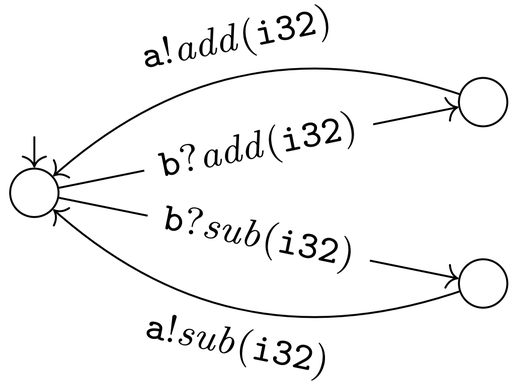
OPTIMISED A



OPTIMISED B

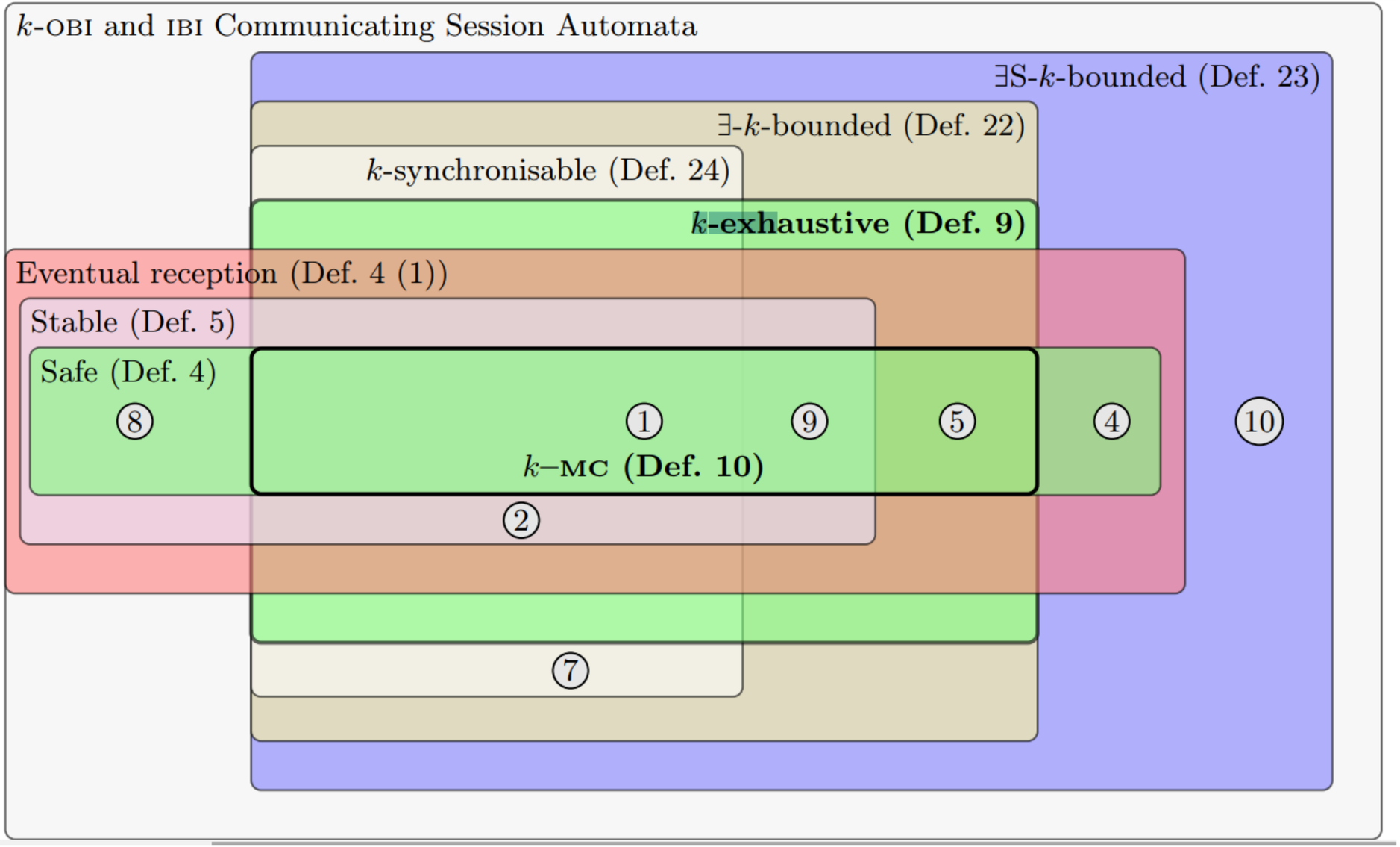


OPTIMISED C



Safe?

k-Multiparty Compatibility [CAV'19]



Asynchronous Subtyping

Existing work

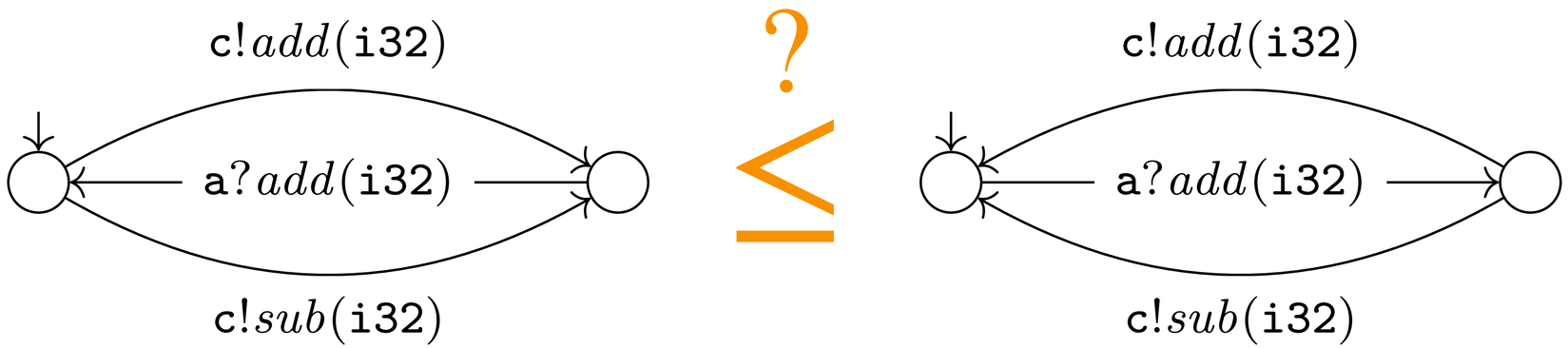
- Relation given by [Ghilezan et al., POPL 2021]
  - Sound ✅
  - Complete ✅
  - Decidable [Lange and NY, FoSSaCs 2017] ❌
- Our aim is a sound and decidable algorithm
- **Theorem [POPL 2021]**

  Internal and external choices can be decomposed into single input and single output trees

Asynchronous Subtyping

The Problem

- Choice and recursion make subtyping hard



Nested Session Asynchronous Subtyping

Precise Subtyping by Chen, Dezani et al

$$\frac{S^r_m \leqslant S_m \quad S^s_m \leqslant S_m \quad S^r_p \leqslant S_p \quad S^s_p \leqslant S_p \quad T_m \leqslant ?\mathsf{r}(S_r).T_r \,\&\, ?\mathsf{s}(S_s).T_s \quad T_p \leqslant ?\mathsf{r}(S_r).T'_r \,\&\, ?\mathsf{s}(S_s).T'_s}{!\mathsf{m}\langle S_m\rangle.T_m \oplus !\mathsf{p}\langle S_p\rangle.T_p \leqslant ?\mathsf{r}(S_r).(!\mathsf{m}\langle S^r_m\rangle.T_r \oplus !\mathsf{p}\langle S^r_p\rangle.T'_r \oplus !\mathsf{q}\langle S_q\rangle.T_q) \,\&\, ?\mathsf{s}(S_s).(!\mathsf{m}\langle S^s_m\rangle.T_s \oplus !\mathsf{p}\langle S^s_p\rangle.T'_s)}$$

Figure 3: Application of [SUB-PERM-ASYNC], where $T_m = ?\mathsf{r}(S_r).T_r \,\&\, ?\mathsf{s}(S_s).T_s \,\&\, ?\mathsf{u}(S_u).T_u$ and $T_p = ?\mathsf{r}(S'_r).T'_r \,\&\, ?\mathsf{s}(S_s).T'_s$ and we assume $S'_r \leqslant S_r$.

$$\mathsf{T}_0 = \mathsf{T}'_0 = \mathsf{end}$$
$$\mathsf{T}_{n+1} = !m.(?r.\mathsf{T}_n \,\&\, ?s.\mathsf{T}_n \,\&\, ?u.\mathsf{T}_n) \oplus !p.(?r.\mathsf{T}_n \,\&\, ?s.\mathsf{T}_n)$$
$$\mathsf{T}'_{n+1} = ?r.(!m.\mathsf{T}'_n \oplus !p.\mathsf{T}'_n \oplus !q.\mathsf{T}'_n) \,\&\, ?s.(!m.\mathsf{T}'_n \oplus !p.\mathsf{T}'_n)$$

Asynchronous Subtyping

SISO Refinement [POPL'21]

SISO trees are just paths — i.e. sequences of inputs and outputs!

$$\frac{}{\mathtt{end} \lesssim \mathtt{end}} \qquad \frac{\mathsf{S}' \leqslant: \mathsf{S} \quad \mathsf{W} \lesssim \mathsf{W}'}{\mathbf{p}?\ell(\mathsf{S}).\mathsf{W} \lesssim \mathbf{p}?\ell(\mathsf{S}').\mathsf{W}'} \qquad \frac{\mathsf{S}' \leqslant: \mathsf{S} \quad \mathsf{W} \lesssim \mathcal{A}^{(\mathbf{p})}.\mathsf{W}' \quad \mathtt{act}(\mathsf{W}) = \mathtt{act}(\mathcal{A}^{(\mathbf{p})}.\mathsf{W}')}{\mathbf{p}?\ell(\mathsf{S}).\mathsf{W} \lesssim \mathcal{A}^{(\mathbf{p})}.\mathbf{p}?\ell(\mathsf{S}').\mathsf{W}'}$$

$$\frac{\mathsf{S} \leqslant: \mathsf{S}' \quad \mathsf{W} \lesssim \mathsf{W}'}{\mathbf{p}!\ell(\mathsf{S}).\mathsf{W} \lesssim \mathbf{p}!\ell(\mathsf{S}').\mathsf{W}'} \qquad \frac{\mathsf{S} \leqslant: \mathsf{S}' \quad \mathsf{W} \lesssim \mathcal{B}^{(\mathbf{p})}.\mathsf{W}' \quad \mathtt{act}(\mathsf{W}) = \mathtt{act}(\mathcal{B}^{(\mathbf{p})}.\mathsf{W}')}{\mathbf{p}!\ell(\mathsf{S}).\mathsf{W} \lesssim \mathcal{B}^{(\mathbf{p})}.\mathbf{p}!\ell(\mathsf{S}').\mathsf{W}'}$$

$$\mathcal{A}^{(\mathbf{p})} ::= \mathbf{q}?\ell(\mathsf{S}) \parallel \mathbf{q}?\ell(\mathsf{S}).\mathcal{A}^{(\mathbf{p})} \qquad \mathcal{B}^{(\mathbf{p})} ::= \mathbf{r}?\ell(\mathsf{S}) \parallel \mathbf{q}!\ell(\mathsf{S}) \parallel \mathbf{r}?\ell(\mathsf{S}).\mathcal{B}^{(\mathbf{p})} \parallel \mathbf{q}!\ell(\mathsf{S}).\mathcal{B}^{(\mathbf{p})} \quad (\mathbf{q} \neq \mathbf{p})$$

$$\frac{\forall \mathsf{U}' \in [\![\mathbb{T}']\!]_{\mathsf{so}} \quad \forall \mathsf{V} \in [\![\mathbb{T}]\!]_{\mathsf{si}} \quad \exists \mathsf{W}' \in [\![\mathsf{U}']\!]_{\mathsf{si}} \quad \exists \mathsf{W} \in [\![\mathsf{V}]\!]_{\mathsf{so}} \qquad \mathsf{W}' \lesssim \mathsf{W}}{\mathbb{T}' \leqslant \mathbb{T}}$$

Algorithm for Asynchronous Subtyping

Practical, Sound and Terminating

1. Bound the number of times we unroll recursions
2. Only unwrap choice on demand

Asynchronous Subtyping

Session Type Prefix

$$
\begin{array}{rcll}
\pi, \rho & ::= & \epsilon & \text{empty prefix} \\
& | & \mathtt{p}!\ell(S) & \text{message send} \\
& | & \mathtt{p}?\ell(S) & \text{message receive} \\
& | & \pi_1.\pi_2 & \text{concatenation}
\end{array}
$$

Asynchronous Subtyping

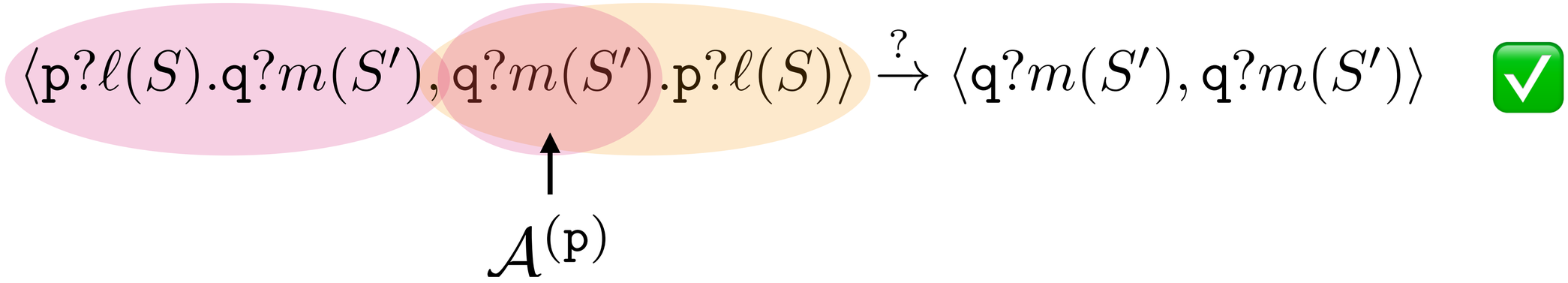
Reduction Rules

$$\mathcal{A}^{(\mathtt{p})} \;::=\; \mathtt{q}?\ell(S) \mid \mathtt{q}?\ell(S).\mathcal{A}^{(\mathtt{p})} \qquad (\mathtt{p} \neq \mathtt{q})$$

$$\frac{S' \leq: S}{\left\langle \mathtt{p}?\ell(S).\pi, \mathcal{A}^{(\mathtt{p})}.\mathtt{p}?\ell(S').\pi' \right\rangle \to \left\langle \pi, \mathcal{A}^{(\mathtt{p})}.\pi' \right\rangle}\ [\textsc{red-}\mathcal{A}]$$

Asynchronous Subtyping

Reduction Rules



Asynchronous Subtyping

Reduction Rules

$$\langle \mathtt{p}?\ell(S).\mathtt{q}?m(S'), \mathtt{q}?m(S').\mathtt{p}?\ell(S)\rangle \xrightarrow{?} \langle \mathtt{q}?m(S'), \mathtt{q}?m(S')\rangle$$ ✅

$$\langle \mathtt{p}?\ell(S).\mathtt{p}?m(S'), \mathtt{p}?m(S').\mathtt{p}?\ell(S)\rangle \xrightarrow{?} \langle \mathtt{p}?m(S'), \mathtt{p}?m(S')\rangle$$ ❌

$\mathcal{A}^{(\mathtt{p})}$

$$\mathcal{A}^{(\mathtt{p})} ::= \mathtt{q}?\ell(S) \mid \mathtt{q}?\ell(S).\mathcal{A}^{(\mathtt{p})} \qquad (\mathtt{p} \neq \mathtt{q})$$

Asynchronous Subtyping

Reduction Rules

$$\mathcal{B}^{(\mathtt{p})} \;::=\; \mathtt{r}?\ell(S) \mid \mathtt{q}!\ell(S) \mid \mathtt{r}?\ell(S).\mathcal{B}^{(\mathtt{p})} \mid \mathtt{q}!\ell(S).\mathcal{B}^{(\mathtt{p})} \qquad (\mathtt{p} \neq \mathtt{q})$$

$$\frac{S' \leq: S}{\left\langle \mathtt{p}!\ell(S).\pi, \mathcal{B}^{(\mathtt{p})}.\mathtt{p}!\ell(S').\pi' \right\rangle \to \left\langle \pi, \mathcal{B}^{(\mathtt{p})}.\pi' \right\rangle} \; [\textsc{red}\text{-}\mathcal{B}]$$

Asynchronous Subtyping

Reduction Rules

$$\mathcal{B}^{(\mathtt{p})} \;::=\; \mathtt{r}?\ell(S) \mid \mathtt{q}!\ell(S) \mid \mathtt{r}?\ell(S).\mathcal{B}^{(\mathtt{p})} \mid \mathtt{q}!\ell(S).\mathcal{B}^{(\mathtt{p})} \qquad (\mathtt{p} \neq \mathtt{q})$$

$$\frac{S' \leq: S}{\left\langle \mathtt{p}!\ell(S).\pi, \mathcal{B}^{(\mathtt{p})}.\mathtt{p}!\ell(S').\pi' \right\rangle \to \left\langle \pi, \mathcal{B}^{(\mathtt{p})}.\pi' \right\rangle} \left[\text{RED-}\mathcal{B}\right]$$

Asynchronous Subtyping

Reduction Rules

$$\mathcal{B}^{(\mathtt{p})} \;::=\; \mathtt{r}?\ell(S) \mid \mathtt{q}!\ell(S) \mid \mathtt{r}?\ell(S).\mathcal{B}^{(\mathtt{p})} \mid \mathtt{q}!\ell(S).\mathcal{B}^{(\mathtt{p})} \qquad (\mathtt{p} \neq \mathtt{q})$$

$$\frac{S' \leq: S}{\left\langle \mathtt{p}!\ell(S).\pi, \mathcal{B}^{(\mathtt{p})}.\mathtt{p}!\ell(S').\pi' \right\rangle \to \left\langle \pi, \mathcal{B}^{(\mathtt{p})}.\pi' \right\rangle} \left[\text{RED-}\mathcal{B}\right]$$

Asynchronous Subtyping

Reduction Rules

$$\mathcal{B}^{(\mathtt{p})} \;::=\; \mathtt{r}?\ell(S) \mid \mathtt{q}!\ell(S) \mid \mathtt{r}?\ell(S).\mathcal{B}^{(\mathtt{p})} \mid \mathtt{q}!\ell(S).\mathcal{B}^{(\mathtt{p})} \qquad (\mathtt{p} \neq \mathtt{q})$$

$$\frac{S' \leq: S}{\left\langle \mathtt{p}!\ell(S).\pi, \mathcal{B}^{(\mathtt{p})}.\mathtt{p}!\ell(S').\pi' \right\rangle \to \left\langle \pi, \mathcal{B}^{(\mathtt{p})}.\pi' \right\rangle} \left[\text{RED-}\mathcal{B}\right]$$

Asynchronous Subtyping

Reduction Rules

$$\langle \mathtt{p}!\ell(S).\mathtt{p}?m(S'), \mathtt{p}?m(S').\mathtt{p}!\ell(S)\rangle \xrightarrow{?} \langle \mathtt{p}?m(S'), \mathtt{p}?m(S')\rangle$$

Asynchronous Subtyping

Reduction Rules

$$\langle \mathtt{p}!\ell(S).\mathtt{p}?m(S'), \mathtt{p}?m(S').\mathtt{p}!\ell(S)\rangle \xrightarrow{?} \langle \mathtt{p}?m(S'), \mathtt{p}?m(S')\rangle$$

Asynchronous Subtyping

Reduction Rules

$$\langle \mathtt{p}!\ell(S).\mathtt{p}?m(S'), \mathtt{p}?m(S').\mathtt{p}!\ell(S)\rangle \xrightarrow{?} \langle \mathtt{p}?m(S'), \mathtt{p}?m(S')\rangle$$

Asynchronous Subtyping

Reduction Rules

$$\langle \mathtt{p}!\ell(S).\mathtt{p}?m(S'), \mathtt{p}?m(S').\mathtt{p}!\ell(S)\rangle \xrightarrow{?} \langle \mathtt{p}?m(S'), \mathtt{p}?m(S')\rangle$$

$$\mathcal{B}^{(\mathtt{p})}$$

Asynchronous Subtyping

Reduction Rules

$$\langle \mathtt{p}!\ell(S).\mathtt{p}?m(S'), \mathtt{p}?m(S').\mathtt{p}!\ell(S)\rangle \xrightarrow{?} \langle \mathtt{p}?m(S'), \mathtt{p}?m(S')\rangle$$

$\mathcal{B}^{(\mathtt{p})}$

✅

Theorems

Termination, Soundness & Complexity

Lemma 3. *Given finite prefixes π and π', $\langle \pi \parallel \pi' \rangle$ can be reduced only a finite number of times.*

Theorem 4 (Termination)**.** *Our subtyping algorithm always eventually terminates.*

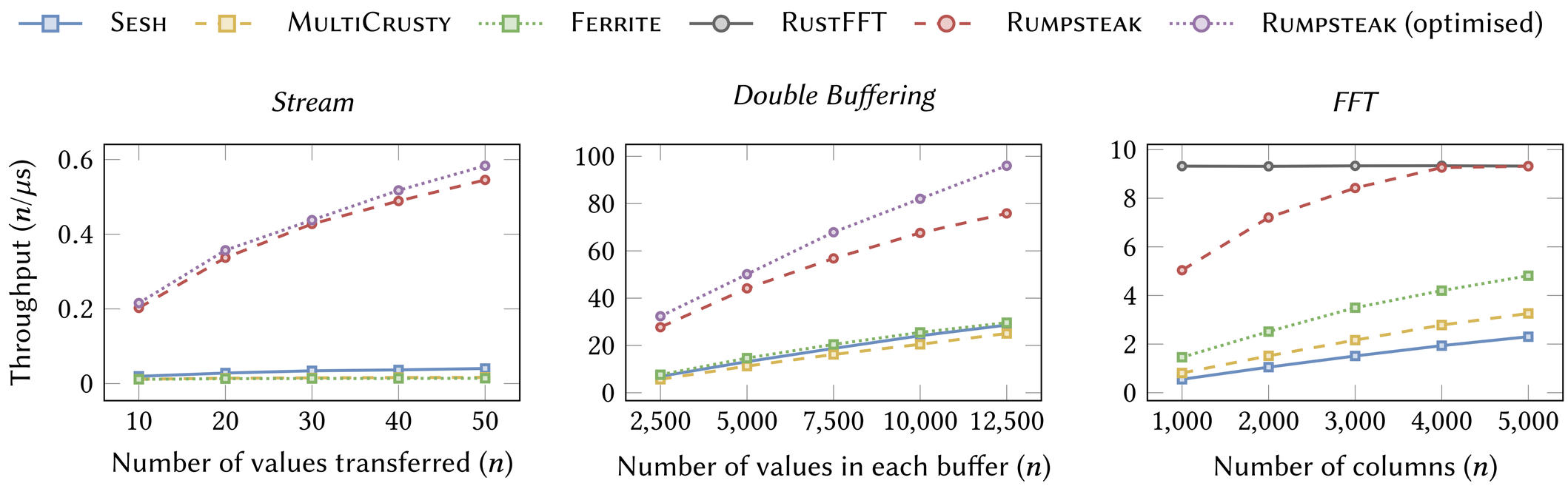
Theorem 5 (Soundness)**.** *Our subtyping algorithm is sound.*

Lemma 6. *Given finite prefixes π and π', the time complexity of reducing $\langle \pi \parallel \pi' \rangle$ is $O(\min(|\pi|, |\pi'|))$.*

Theorem 7 (Complexity)**.** *Consider* T *and* T' *as (possibly infinite) trees* $\mathcal{T}(\mathsf{T})$ *and* $\mathcal{T}(\mathsf{T}')$ *with* asymptotic branching factors b *and* b' *respectively. Our algorithm has time complexity* $O(n \min(b, b')^n)$ *and space complexity* $O(n \min(b, b'))$ *in the worst case to determine if* $\mathsf{T} \leq \mathsf{T}'$ *with bound* n.

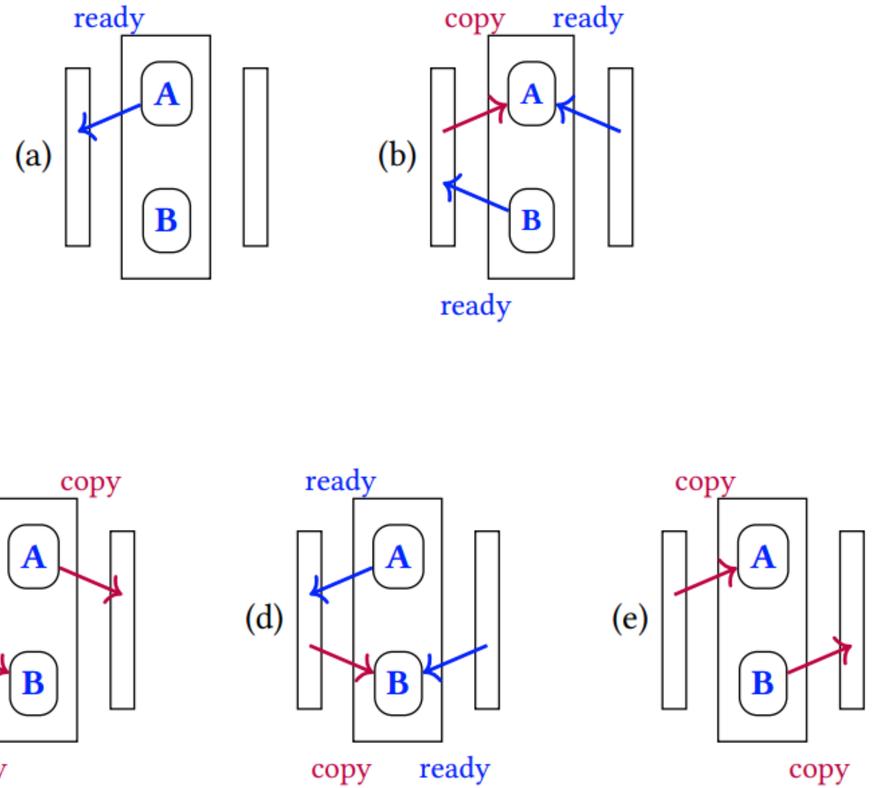
Evaluation

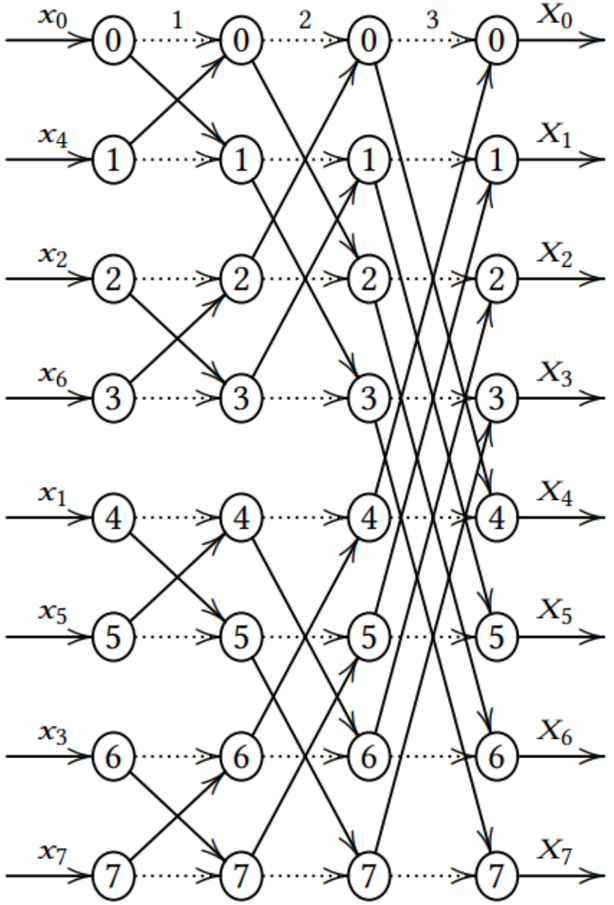
Rust Framework Benchmarks



16-core AMD OpteronTM 6200 Series CPU @ 2.6GHz with hyperthreading, 128GB of RAM, Ubuntu 18.04.5 LTS and Rust Nightly 2021-07-06.
We use version 0.3.5 of the Criterion.rs library and a multi-threaded asynchronous runtime from version 1.11.0 of the Tokio library.

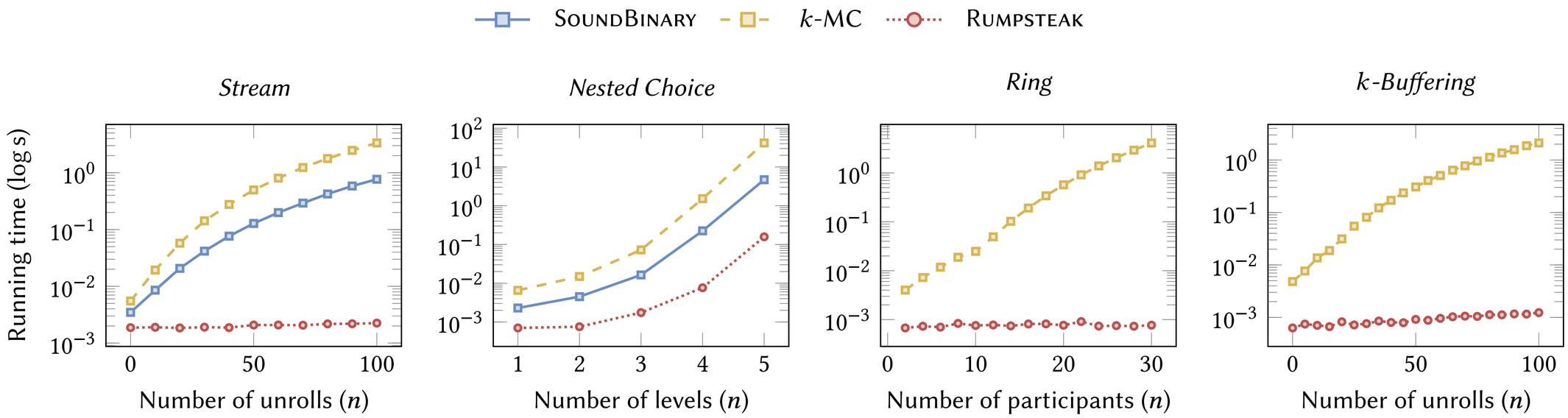
Double DB & Butterfly Topologies for FFT





Evaluation

Asynchronous Reordering Benchmarks



Nested Session Asynchronous Subtyping

Precise Subtyping by Chen, Dezani et al

$$\frac{S^r_m \leqslant S_m \quad S^s_m \leqslant S_m \quad S^r_p \leqslant S_p \quad S^s_p \leqslant S_p \quad T_m \leqslant ?\mathsf{r}(S_r).T_r \,\&\, ?\mathsf{s}(S_s).T_s \quad T_p \leqslant ?\mathsf{r}(S_r).T'_r \,\&\, ?\mathsf{s}(S_s).T'_s}{!\mathsf{m}\langle S_m\rangle.T_m \oplus !\mathsf{p}\langle S_p\rangle.T_p \leqslant ?\mathsf{r}(S_r).(!\mathsf{m}\langle S^r_m\rangle.T_r \oplus !\mathsf{p}\langle S^r_p\rangle.T'_r \oplus !\mathsf{q}\langle S_q\rangle.T_q) \,\&\, ?\mathsf{s}(S_s).(!\mathsf{m}\langle S^s_m\rangle.T_s \oplus !\mathsf{p}\langle S^s_p\rangle.T'_s)}$$

Figure 3: Application of [SUB-PERM-ASYNC], where $T_m = ?\mathsf{r}(S_r).T_r \,\&\, ?\mathsf{s}(S_s).T_s \,\&\, ?\mathsf{u}(S_u).T_u$ and $T_p = ?\mathsf{r}(S'_r).T'_r \,\&\, ?\mathsf{s}(S_s).T'_s$ and we assume $S'_r \leqslant S_r$.

$$\mathsf{T}_0 = \mathsf{T}'_0 = \mathsf{end}$$
$$\mathsf{T}_{n+1} = !m.(?r.\mathsf{T}_n \,\&\, ?s.\mathsf{T}_n \,\&\, ?u.\mathsf{T}_n) \oplus !p.(?r.\mathsf{T}_n \,\&\, ?s.\mathsf{T}_n)$$
$$\mathsf{T}'_{n+1} = ?r.(!m.\mathsf{T}'_n \oplus !p.\mathsf{T}'_n \oplus !q.\mathsf{T}'_n) \,\&\, ?s.(!m.\mathsf{T}'_n \oplus !p.\mathsf{T}'_n)$$

Evaluation

Expressiveness

Protocol	n	AMR	Sesh	Ferrite	MultiCrusty	Rumpsteak	k-MC	SoundBinary
Two Adder	2		✔	✔	✔	✔	✔	✔
Three Adder	3		✗	✗	✔	✔	✔	✗
Stream	2		✔	✔	✔	✔	✔	✔
Optimised Stream	2	✔	✗	✗	✗	✔	✔	✔
Ring	3		✗	✗	✔	✔	✔	✗
Optimised Ring	3	✔	✗	✗	✗	✔	✔	✗
Ring With Choice	3		✗	✗	✔	✔	✔	✗
Optimised Ring With Choice	3	✔	✗	✗	✗	✔	✔	✗
Double Buffering	3		✗	✗	✔	✔	✔	✗
Optimised Double Buffering	3	✔	✗	✗	✗	✔	✔	✗
Alternating Bit	2		✗	✗	✗	✔	✔	✔
Elevator	3	✔	✗	✗	✗	✔	✔	✗
FFT	8		✗	✗	✔	✔	✔	✗
Optimised FFT	8	✔	✗	✗	✗	✔	✔	✗
Authentication	3		✗	✗	✔	✔	✔	✗
Client-Server Log	3		✗	✗	✔	✔	✔	✗
Hospital	2	✔	✗	✗	✗	✗	✗	✔

n Number of participants **AMR** Asynchronous message reordering

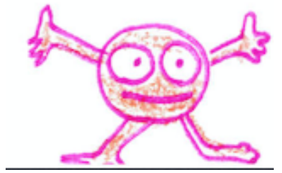
✔ Expressible ✗ Expressible using endpoint types (but without deadlock-freedom guarantee) ✗ Not expressible

References

Multiparty Session Types and Rust

- Multiparty session types and communicating automata
  - Invited paper in the FCT ’21 proceedings
  - Scribble **https://github.com/scribble**
  -  **https://github.com/nuscr**
  - rumpsteak **https://github.com/zakcutner/rumpsteak**
- multi-crusty **http://mrg.doc.ic.ac.uk/tools/multicrusty/**
  - [ECOOP'22] **N. Lagaillardie (IC),** R. Neykova (Brunel), NY



Thank you! Questions?

http://mrg.doc.ic.ac.uk/