Multiparty GV

Functional Multiparty Session Types With Certified Deadlock Freedom

Jules Jacobs
Radboud University

Stephanie Balzer
Carnegie Mellon University

Robbert Krebbers
Radboud University

Usual message passing:

- Stream of messages of fixed type
- e.g., Go, Rust: `Receiver<T>, Sender<T>`

Usual message passing:

- Stream of messages of fixed type
- e.g., Go, Rust: `Receiver<T>, Sender<T>`

Session types:

- Flexible message passing protocols
- Type of message can depend on the state of the protocol
- Linear types

Binary session types: Honda et al. '93, '98

$$c \; : \; \mathbf{!}\mathit{Nat}.\,\mathbf{?}\mathit{Bool}.\,\mathbf{!}\mathit{Nat}.\,\mathsf{End}$$

Binary session types: Honda et al. '93, '98

$$c \; : \; !Nat.?Bool.!Nat.\mathrm{End}$$

$$\Updownarrow \text{dual}$$

$$c' : \; ?Nat.!Bool.?Nat.\mathrm{End}$$

Binary session types: Honda et al. ’93, ’98

$$c \;:\; \mathbf{!}\mathit{Nat}.\,\mathbf{?}\mathit{Bool}.\,\mathbf{!}\mathit{Nat}.\,\mathsf{End}$$

$$\Updownarrow \text{dual}$$

$$c' : \mathbf{?}\mathit{Nat}.\,\mathbf{!}\mathit{Bool}.\,\mathbf{?}\mathit{Nat}.\,\mathsf{End}$$

$$s ::= \underbrace{\mathbf{!}\tau.s \mid \mathbf{?}\tau.s}_{\text{send/recv}} \mid \mathsf{End}$$

Binary session types: Honda et al. ’93, ’98

$$c \;:\; !\mathit{Nat}.\,?\mathit{Bool}.\,!\mathit{Nat}.\,\mathsf{End}$$

$$\Updownarrow \text{dual}$$

$$c' : \;?\mathit{Nat}.\,!\mathit{Bool}.\,?\mathit{Nat}.\,\mathsf{End}$$

$$s ::= \underbrace{!\tau.s \mid ?\tau.s}_{\text{send/recv}} \mid \mathsf{End} \mid \underbrace{s \oplus s \mid s \,\&\, s}_{\text{choice}}$$

Binary session types: Honda et al. ’93, ’98

$$c \;:\; !\mathit{Nat}.\,?\mathit{Bool}.\,!\mathit{Nat}.\,\mathsf{End}$$

$$\Updownarrow \text{dual}$$

$$c' : \;?\mathit{Nat}.\,!\mathit{Bool}.\,?\mathit{Nat}.\,\mathsf{End}$$

$$s ::= \underbrace{!\tau.s \mid ?\tau.s}_{\text{send/recv}} \mid \mathsf{End} \mid \underbrace{s \oplus s \mid s \mathbin{\&} s}_{\text{choice}} \mid \underbrace{\mu x.s \mid x}_{\text{recursion}}$$

Binary session types: Honda et al. '93, '98

$$c \;:\; !Nat.\,?Bool.\,!Nat.\,\mathsf{End}$$

$$\Updownarrow \text{dual}$$

$$c' \;:\; ?Nat.\,!Bool.\,?Nat.\,\mathsf{End}$$

$$s ::= \underbrace{!\tau.s \mid ?\tau.s}_{\text{send/recv}} \mid \mathsf{End} \mid \underbrace{s \oplus s \mid s \,\&\, s}_{\text{choice}} \mid \underbrace{\mu x.s \mid x}_{\text{recursion}}$$

$$\tau ::= Nat \mid Bool \mid \tau \times \tau \mid \tau \to \tau \mid \ldots$$

Binary session types: Honda et al. '93, '98

$$c \;:\; !Nat.\,?Bool.\,!Nat.\,\mathrm{End}$$

$$\Updownarrow \text{dual}$$

$$c' : \;?Nat.\,!Bool.\,?Nat.\,\mathrm{End}$$

$$s ::= \underbrace{!\tau.s \mid ?\tau.s}_{\text{send/recv}} \mid \mathrm{End} \mid \underbrace{s \oplus s \mid s \,\&\, s}_{\text{choice}} \mid \underbrace{\mu x.s \mid x}_{\text{recursion}}$$

$$\tau ::= Nat \mid Bool \mid \tau \times \tau \mid \tau \to \tau \mid \ldots \mid \underbrace{s}_{\text{first-class channels}}$$

Multiparty session types: Honda et al. '08

$$\left.\begin{array}{ll} c_0 : & !^1 Nat.\,?^2 Nat.\,\mathsf{End} \\ c_1 : & ?^0 Nat.\,!^2 Bool.\,\mathsf{End} \\ c_2 : & ?^1 Bool.\,!^0 Nat.\,\mathsf{End} \end{array}\right\}\ \text{consistent}$$

Two worlds of session types

GV family languages
Gay, Vasconcelos '10, Wadler '12

- Binary session types

MPST family languages
Honda '08

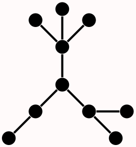
- Multiparty session types

Two worlds of session types

GV family languages

Gay, Vasconcelos '10, Wadler '12

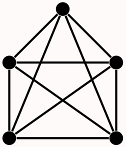
- Binary session types
- Deadlock-freedom by duality & linear typing



MPST family languages

Honda '08

- Multiparty session types
- Deadlock-freedom by global consistency check

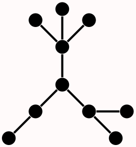


Two worlds of session types

GV family languages

Gay, Vasconcelos '10, Wadler '12

- Binary session types
- Deadlock-freedom by duality & linear typing

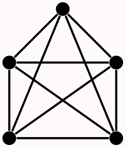


- Dynamic spawning

MPST family languages

Honda '08

- Multiparty session types
- Deadlock-freedom by global consistency check

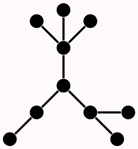


- One static session

Two worlds of session types

GV family languages
Gay, Vasconcelos ’10, Wadler ’12

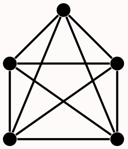
- Binary session types
- Deadlock-freedom by duality & linear typing



- Dynamic spawning
- Channels first class values

MPST family languages
Honda ’08

- Multiparty session types
- Deadlock-freedom by global consistency check



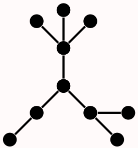
- One static session
- Channels second class

Two worlds of session types

GV family languages

Gay, Vasconcelos ’10, Wadler ’12

- Binary session types
- Deadlock-freedom by duality & linear typing

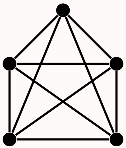


- Dynamic spawning
- Channels first class values
- Functional programming

MPST family languages

Honda ’08

- Multiparty session types
- Deadlock-freedom by global consistency check

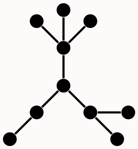


- One static session
- Channels second class
- Pi calculus variants

Two worlds of session types

GV family languages
Gay, Vasconcelos '10, Wadler '12

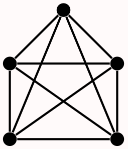
- Binary session types
- Deadlock-freedom by duality & linear typing



- Dynamic spawning
- Channels first class values
- Functional programming

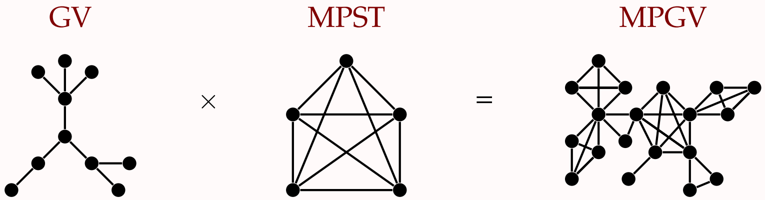
MPST family languages
Honda '08

- Multiparty session types
- Deadlock-freedom by global consistency check

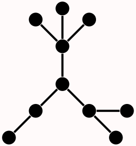


- One static session
- Channels second class
- Pi calculus variants

GV $\times$ MPST = MPGV

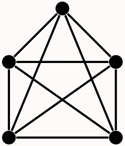

GV
×
MPST
=
MPGV

GV



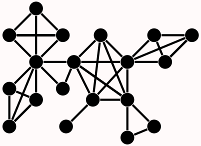
×

MPST



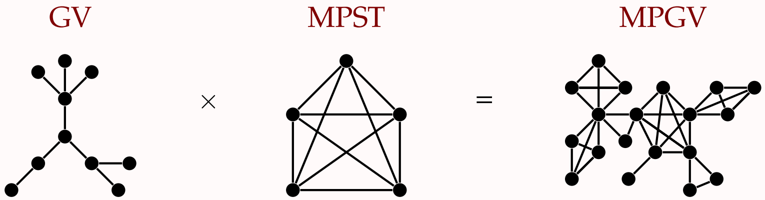
=

MPGV



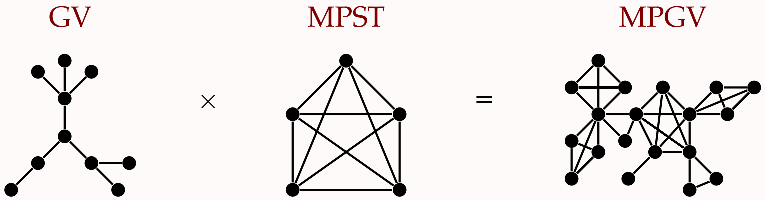
Contributions

1. **Design of MPGV:** concurrent λ-calculus with multiparty message-passing channels as first-class values (+ channel/thread spawning + rectypes + choice)



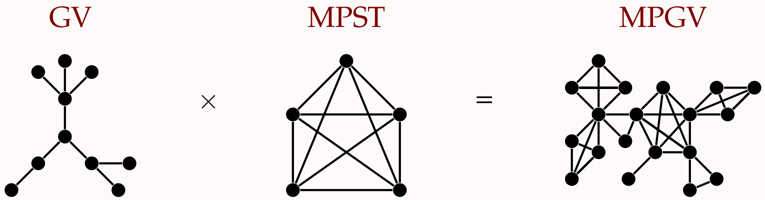
Contributions

1. **Design of MPGV:** concurrent λ-calculus with multiparty message-passing channels as first-class values (+ channel/thread spawning + rectypes + choice)
2. **MPGV ⊃ GV** using new **redirect** for modular programming



Contributions

1. **Design of MPGV:** concurrent λ-calculus with multiparty message-passing channels as first-class values (+ channel/thread spawning + rectypes + choice)
2. **MPGV ⊃ GV** using new **redirect** for modular programming
3. **Key property:** well-typed programs don't get stuck (global progress ⇒ no **receive** deadlocks)



Contributions

1. **Design of MPGV:** concurrent λ-calculus with multiparty message-passing channels as first-class values (+ channel/thread spawning + rectypes + choice)
2. **MPGV ⊃ GV** using new **redirect** for modular programming
3. **Key property:** well-typed programs don't get stuck (global progress ⇒ no **receive** deadlocks)
4. **Meta theory:** fully mechanized in Coq

Tour of MPGV: n-ary fork

Inspired by multi-cut of Carbone et al. CONCUR'16:

$$
\begin{aligned}
&\textbf{let } c_0 = \textbf{fork}(\lambda c_1.\, e_1,\\
&\qquad\qquad\qquad\;\; \lambda c_2.\, e_2,\\
&\qquad\qquad\qquad\;\; \lambda c_3.\, e_3)\\
&\textbf{in } e_0
\end{aligned}
$$

Tour of MPGV: n-ary fork

Inspired by multi-cut of Carbone et al. CONCUR'16:

$$\begin{aligned}&\textbf{let}\ c_0 : s_0 = \textbf{fork}\langle s\rangle\ (\lambda c_1 : s_1.\ e_1 : (),\\ &\qquad\qquad\qquad\qquad \lambda c_2 : s_2.\ e_2 : (),\\ &\qquad\qquad\qquad\qquad \lambda c_3 : s_3.\ e_3 : ())\\ &\textbf{in}\ e_0\end{aligned}$$

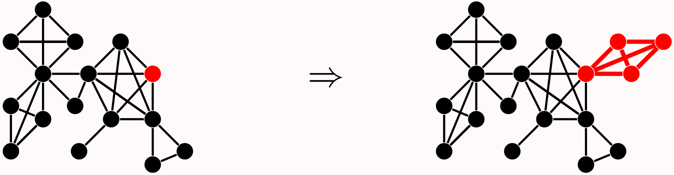
$(s_0, s_1, \ldots, s_n)$ consistent

Tour of MPGV: n-ary fork

Inspired by multi-cut of Carbone et al. CONCUR'16:

$$\begin{aligned}\textbf{let } c_0 : s_0 = \textbf{fork}\langle s\rangle\, (&\lambda c_1 : s_1.\ e_1 : (),\\ &\lambda c_2 : s_2.\ e_2 : (),\\ &\lambda c_3 : s_3.\ e_3 : ())\\ \textbf{in } e_0\end{aligned}$$

$(s_0, s_1, \ldots, s_n)$ consistent



Tour of MPGV: send and receive

$$\textbf{send}^{p} : (!^{p}\tau.\, s) \times \tau \to s$$

$$\textbf{receive}^{p} : (?^{p}\tau.\, s) \to \tau \times s$$

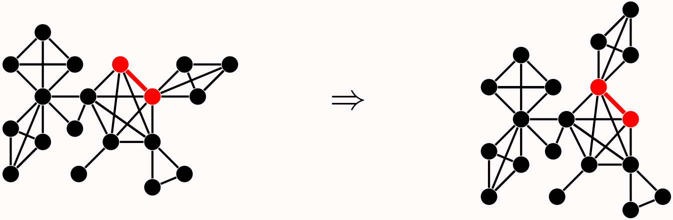
(+ choice)

Tour of MPGV: send and receive

$$\textbf{send}^{p} : (!^{p}\tau.\, s) \times \tau \to s$$
$$\textbf{receive}^{p} : (?^{p}\tau.\, s) \to \tau \times s$$

(+ choice)

- First-class channels:

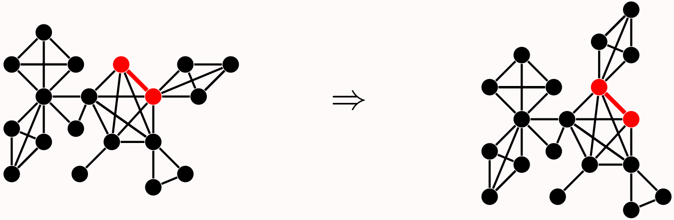


Tour of MPGV: send and receive

$$\textbf{send}^{p} : (!^{p}\tau.\, s) \times \tau \to s$$
$$\textbf{receive}^{p} : (?^{p}\tau.\, s) \to \tau \times s$$

(+ choice)

- First-class channels:



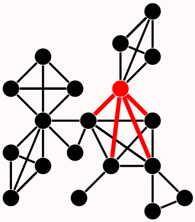
- In the paper: asynchronous semantics (messages go via buffers)

Close

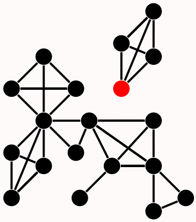
$$\textbf{close} : \mathsf{End} \to (\,)$$

Close

$$\textbf{close} : \mathsf{End} \rightarrow (\,)$$



$\Rightarrow$



Tour of MPGV: redirect

We want: MPGV $\supset$ GV

Tour of MPGV: redirect

We want: MPGV $\supset$ GV

Problem: participant annotations get in the way

$$!^{0} \mathit{Nat}.\,?^{0} \mathit{Bool}.\,\mathsf{End} \neq !^{1} \mathit{Nat}.\,?^{1} \mathit{Bool}.\,\mathsf{End}$$

Tour of MPGV: redirect

We want: MPGV $\supset$ GV
Problem: participant annotations get in the way

$$!^{0}\mathit{Nat}.\,?^{0}\mathit{Bool}.\,\mathsf{End} \neq\, !^{1}\mathit{Nat}.\,?^{1}\mathit{Bool}.\,\mathsf{End}$$

Solution:

$$\textbf{redirect}[1 \mapsto 0] : \quad !^{0}\mathit{Nat}.\,?^{0}\mathit{Bool}.\,\mathsf{End} \rightarrow\, !^{1}\mathit{Nat}.\,?^{1}\mathit{Bool}.\,\mathsf{End}$$

Tour of MPGV: redirect

We want: MPGV $\supset$ GV
Problem: participant annotations get in the way

$$!^{0} \mathit{Nat}. ?^{0} \mathit{Bool}. \mathsf{End} \neq !^{1} \mathit{Nat}. ?^{1} \mathit{Bool}. \mathsf{End}$$

Solution:

$$\textbf{redirect}[1 \mapsto 0] : \; !^{0} \mathit{Nat}. ?^{0} \mathit{Bool}. \mathsf{End} \to !^{1} \mathit{Nat}. ?^{1} \mathit{Bool}. \mathsf{End}$$

- Used in translation from GV to MPGV
- In the paper: useful for modularity

Mechanized meta theory

> Unfortunately, the more complicated the behaviour is, the more *error-prone* the theory becomes. The literature reveals broken proofs of subject reduction for several MPST systems [42], and a flaw of the decidability of subtyping [6] for asynchronous MPST. All of which are caused by an incorrect understanding of the (asynchronous) behaviour of types.

– Castro-Perez et al. PLDI'21

Mechanized meta theory

> Unfortunately, the more complicated the behaviour is, the more *error-prone* the theory becomes. The literature reveals broken proofs of subject reduction for several MPST systems [42], and a flaw of the decidability of subtyping [6] for asynchronous MPST. All of which are caused by an incorrect understanding of the (asynchronous) behaviour of types.

– Castro-Perez et al. PLDI'21

Mechanized in Coq:

- Language definition $\approx$500 LOC, proofs $\approx$10,000 LOC
- Small-step asynchronous semantics with threads and buffers

Mechanized meta theory

> Unfortunately, the more complicated the behaviour is, the more *error-prone* the theory becomes. The literature reveals broken proofs of subject reduction for several MPST systems [42], and a flaw of the decidability of subtyping [6] for asynchronous MPST. All of which are caused by an incorrect understanding of the (asynchronous) behaviour of types.

– Castro-Perez et al. PLDI'21

Mechanized in Coq:

- Language definition ≈500 LOC, proofs ≈10,000 LOC
- Small-step asynchronous semantics with threads and buffers
- **Safety:** threads don't get stuck (except on **receive**)

Mechanized meta theory

> Unfortunately, the more complicated the behaviour is, the more *error-prone* the theory becomes. The literature reveals broken proofs of subject reduction for several MPST systems [42], and a flaw of the decidability of subtyping [6] for asynchronous MPST. All of which are caused by an incorrect understanding of the (asynchronous) behaviour of types.

– Castro-Perez et al. PLDI'21

Mechanized in Coq:

- Language definition ≈500 LOC, proofs ≈10,000 LOC
- Small-step asynchronous semantics with threads and buffers
- **Safety:** threads don't get stuck (except on **receive**)
- **Deadlock freedom:** no subset of threads stuck on each other

Mechanized meta theory

> Unfortunately, the more complicated the behaviour is, the more *error-prone* the theory becomes. The literature reveals broken proofs of subject reduction for several MPST systems [42], and a flaw of the decidability of subtyping [6] for asynchronous MPST. All of which are caused by an incorrect understanding of the (asynchronous) behaviour of types.

– Castro-Perez et al. PLDI'21

Mechanized in Coq:

- Language definition ≈500 LOC, proofs ≈10,000 LOC
- Small-step asynchronous semantics with threads and buffers
- **Safety:** threads don't get stuck (except on **receive**)
- **Deadlock freedom:** no subset of threads stuck on each other
- **Leak freedom:** no messages left behind

How to know that you mechanized the right theorem?

How to know that you mechanized the right theorem?

- The statement deadlock freedom is complex (5 LOC)
  - Statement relies on several auxiliary definitions ($\approx$100 LOC)
  - How do we know that the theorem statement is not wrong?

How to know that you mechanized the right theorem?

- The statement deadlock freedom is complex (5 LOC)
  - Statement relies on several auxiliary definitions ($\approx$100 LOC)
  - How do we know that the theorem statement is not wrong?
- Prove easy to understand corollary, **global progress**:
  If $e : ()$ and $e \leadsto^* \rho$ then either $\rho = \emptyset$ or $\exists \rho'.\ \rho \leadsto \rho'$

How to know that you mechanized the right theorem?

- The statement deadlock freedom is complex (5 LOC)
  - Statement relies on several auxiliary definitions ($\approx$100 LOC)
  - How do we know that the theorem statement is not wrong?
- Prove easy to understand corollary, **global progress**:
  If $e : ()$ and $e \rightsquigarrow^* \rho$ then either $\rho = \emptyset$ or $\exists \rho'.\ \rho \rightsquigarrow \rho'$
- 1 LOC, no auxiliary definitions
- Sanity check on deadlock freedom statement

What’s in the paper

- Full MGPV language definition and semantics
  - Asynchronous semantics with buffers
  - Choice in session types
  - Recursive types & recursive session types (mutual)
  - Linear & unrestricted types

What's in the paper

- Full MGPV language definition and semantics
  - Asynchronous semantics with buffers
  - Choice in session types
  - Recursive types & recursive session types (mutual)
  - Linear & unrestricted types
- Multiparty consistency & global types

What's in the paper

- Full MGPV language definition and semantics
  - Asynchronous semantics with buffers
  - Choice in session types
  - Recursive types & recursive session types (mutual)
  - Linear & unrestricted types
- Multiparty consistency & global types
- Deadlock freedom proof
  - With pictures
  - Formal details (with separation logic)

What's in the paper

- Full MGPV language definition and semantics
  - Asynchronous semantics with buffers
  - Choice in session types
  - Recursive types & recursive session types (mutual)
  - Linear & unrestricted types
- Multiparty consistency & global types
- Deadlock freedom proof
  - With pictures
  - Formal details (with separation logic)
- Encoding GV in MPGV

What's in the paper

- Full MGPV language definition and semantics
  - Asynchronous semantics with buffers
  - Choice in session types
  - Recursive types & recursive session types (mutual)
  - Linear & unrestricted types
- Multiparty consistency & global types
- Deadlock freedom proof
  - With pictures
  - Formal details (with separation logic)
- Encoding GV in MPGV
- All theorems mechanized in Coq

What's in the paper

- Full MGPV language definition and semantics
  - Asynchronous semantics with buffers
  - Choice in session types
  - Recursive types & recursive session types (mutual)
  - Linear & unrestricted types
- Multiparty consistency & global types
- Deadlock freedom proof
  - With pictures
  - Formal details (with separation logic)
- Encoding GV in MPGV
- All theorems mechanized in Coq

Questions?