Certifying and Signing Data in Multiparty Session Types

BETTY Meeting

Bernardo Toninho, Nobuko Yoshida

18 March 2016
Introduction

Context

- Multiparty session types [Honda et al., 08-]
  - Specify coordination of multiple communicating agents.
  - **Global types** specify interactions involving multiple peers,
  - Automatically mapped to **local** types,
  - Checked against individual endpoint processes.
Multiparty session types [Honda et al., 08-]
- Specify coordination of multiple communicating agents.
- **Global types** specify interactions involving multiple peers,
- Automatically mapped to local types,
- Checked against individual endpoint processes.

Value-dependent session types [Toninho et al., 2011]
- Extend binary session types with dependent data I/O.
- Types specify interaction + properties of exchanged data.
- Properties are witnessed by explicit proofs.
- Type safety entails comm. safety + property satisfaction.
Introduction

Trip to Malta

\[ G_{BM} = B \rightarrow \text{AirMalta} : (\text{String}, \text{Int}). \quad \text{Destination } + \text{ date} \]

AirMalta \rightarrow B : (\text{String}). \quad \text{Flight Info.}
Introduction

Trip to Malta

\[ G_{BM} = B \rightarrow \text{AirMalta} : (\text{String}, \text{Int}). \quad \text{Destination + date} \]
\[ \text{AirMalta} \rightarrow B : (\text{String}). \quad \text{Flight Info.} \]
\[ B \rightarrow \text{Booking} : (\text{String}, \text{Int}). \quad \text{Destination + date} \]
Introduction

Trip to Malta

\[ G_{BM} = B \rightarrow \text{AirMalta} : (\text{String}, \text{Int}). \quad \text{Destination + date} \\
\text{AirMalta} \rightarrow B : (\text{String}). \quad \text{Flight Info.} \\
B \rightarrow \text{Booking} : (\text{String}, \text{Int}). \quad \text{Destination + date} \\
\text{Booking} \rightarrow B : (\text{List String}). \quad \text{Options} \\
B \rightarrow \text{Booking} : (\text{String}). \quad \text{Choose} \\
\text{Booking} \rightarrow B : (\text{Int}). \quad \text{Receipt} \]
Introduction

Trip to Malta

\[ G_{BM} = \begin{align*}
B & \rightarrow \text{AirMalta} : (\text{String, Int}). & \text{Destination + date} \\
\text{AirMalta} & \rightarrow B : (\text{String}). & \text{Flight Info.} \\
B & \rightarrow \text{Booking} : (\text{String, Int}). & \text{Destination + date} \\
\text{Booking} & \rightarrow B : (\text{List String}). & \text{Options} \\
B & \rightarrow \text{Booking} : (\text{String}). & \text{Choose} \\
\text{Booking} & \rightarrow B : (\text{Int}). & \text{Receipt} \\
B & \rightarrow \text{COST} : (\text{String, Int}). & \text{Expense claims} \\
\text{ICClaims} & \rightarrow B : (\text{ok : end; nok : end}) & \text{ok or not}
\end{align*} \]
Introduction

Trip to Malta

\[ G_{BM} = B \rightarrow \text{AirMalta} : (\text{String}, \text{Int}). \text{Destination} + \text{date} \\
\text{AirMalta} \rightarrow B : (\text{String}). \text{Flight Info.} \\
B \rightarrow \text{Booking} : (\text{String}, \text{Int}). \text{Destination} + \text{date} \\
\text{Booking} \rightarrow B : (\text{List} \text{String}). \text{Options} \\
B \rightarrow \text{Booking} : (\text{String}). \text{Choose} \\
\text{Booking} \rightarrow B : (\text{Int}). \text{Receipt} \\
B \rightarrow \text{COST} : (\text{String}, \text{Int}). \text{Expense claims} \\
\text{ICClaims} \rightarrow B : (\text{ok : end}; \text{nok : end}) \text{ok or not} \]

- Type specifies the communication between the parties.
- The majority of the functionality is absent from the type.
Introduction

Motivation

- MPST ensure communication safety in multi-agent protocols.
- Do not capture functional constraints of protocols.
Introduction

Motivation

- MPST ensure communication safety in multi-agent protocols.
- Do not capture functional constraints of protocols.

Our Proposal

- A session type discipline for certifying **global** properties of exchanged data, based on value dependent MPST.
- Ensures adherence to session discipline,
- + rich constraint satisfaction based on explicit runtime witnesses.
Certifying Data in MPST

Revisiting the trip to Malta

$$G_{BM} = B \rightarrow \text{AirMalta} : (d_1:\text{String}, d_2:\text{Int}). \quad \text{Dest. + date}$$

$$\text{AirMalta} \rightarrow B : (f:\text{String}). \quad \text{Flight Info.}$$

$$B \rightarrow \text{Booking} : (\text{String}(d_1), \text{Int}(d_2)). \quad \text{Dest. + date}$$
Certifying Data in MPST

Revisiting the trip to Malta

\[ G_{BM} = \begin{align*} 
  B & \rightarrow \text{AirMalta} : (d_1:\text{String}, d_2:\text{Int}). & \text{Dest. + date} \\
  \text{AirMalta} & \rightarrow B : (f:\text{String}). & \text{Flight Info.} \\
  B & \rightarrow \text{Booking} : (\text{String}(d_1), \text{Int}(d_2)). & \text{Dest. + date} \\
  \text{Booking} & \rightarrow B : (\text{opt}:\text{List String}). & \text{Options} \\
  B & \rightarrow \text{Booking} : (\Sigma s:\text{String}. s \in \text{opt}). & \text{Choose} \\
  \text{Booking} & \rightarrow B : (r:\text{Int}). & \text{Receipt} 
\end{align*} \]
Certifying Data in MPST

Revisiting the trip to Malta

\[ G_{BM} = B \rightarrow \text{AirMalta} : (d_1: \text{String}, d_2: \text{Int}). \quad \text{Dest. + date} \\
\text{AirMalta} \rightarrow B : (f: \text{String}). \quad \text{Flight Info.} \\
B \rightarrow \text{Booking} : (\text{String}(d_1), \text{Int}(d_2)). \quad \text{Dest. + date} \\
\text{Booking} \rightarrow B : (\text{opt}: \text{List String}). \quad \text{Options} \\
B \rightarrow \text{Booking} : (\Sigma s: \text{String}. s \in \text{opt}). \quad \text{Choose} \\
\text{Booking} \rightarrow B : (r: \text{Int}). \quad \text{Receipt} \\
B \rightarrow \text{COST} : (\text{String}(f), \text{Int}(r)). \quad \text{Expense claims} \\
\text{ICClaims} \rightarrow B : (\text{ok} : \text{end}; \text{nok} : \text{end}) \quad \text{ok or not} \]

- Type families and dependent tuples allow for a richer spec.
- Singleton types (e.g. \text{String}(d)) specify equality constraints.
Certifying Data in MPST

Types

Global and Message Types

\[
G ::= p \to q : (x: \tau).G \\
| p \to q : (l_j: G_j)_{j \in J} \\
| p \to q : (T).G \\
| \mu t (x = M: \tau).G \mid t\langle M \rangle \\
| \text{end}
\]

\[
\tau, \sigma ::= 
\]
### Global and Message Types

\[
G ::= p \rightarrow q : (x:\tau).G \\
| p \rightarrow q : (l_j:G_j)_{j \in J} \\
| p \rightarrow q : (T).G \\
| \mu t \ (x = M:\tau).G \mid t\langle M \rangle \\
| \text{end}
\]

\[
\tau, \sigma ::= \Pi x:\tau.\sigma
\]
Certifying Data in MPST

Types

Global and Message Types

\[ G ::= p \to q : (x:\tau).G \]

| \[ p \to q : (l_j:G_j)_{j \in J} \] |
| \[ p \to q : (T).G \] |
| \[ \mu t \ (x = M:\tau).G \mid t\langle M\rangle \] |
| \[ \text{end} \] |

\[ \tau, \sigma ::= \Pi x:\tau.\sigma \mid \Sigma x:\tau.\sigma \mid b \]
### Global and Message Types

\[
G ::= p \rightarrow q : (x:τ).G \\
| p \rightarrow q : (l_j:G_j)_{j \in J} \\
| p \rightarrow q : (T).G \\
| \mu t (x = M:τ).G | t\langle M\rangle \\
| \text{end}
\]

\[
\tau, \sigma ::= \Pi x:τ.σ | \Sigma x:τ.σ | b | S(M)
\]
Global and Message Types

\[ G ::= p \to q : (x:\tau).G \]
\[ \mid p \to q : (l_j:G_j)_{j \in J} \]
\[ \mid p \to q : (T).G \]
\[ \mid \mu t \ (x = M:\tau).G \mid t\langle M\rangle \]
\[ \mid \text{end} \]

\[ \tau, \sigma ::= \Pi x:\tau.\sigma \mid \Sigma x:\tau.\sigma \mid b \mid S(M) \]

Singleton Types

- \( S(M) \), where \( M : b \).
- \( S(M) \) denotes a value of type \( b \) equal to \( M \).
Certifying Data in MPST

Types

Global and Message Types

\[ G ::= p \rightarrow q : (x:\tau).G \]
\[ \mid p \rightarrow q : (l_j:G_j)_{j \in J} \]
\[ \mid p \rightarrow q : (T).G \]
\[ \mid \mu t \, (x = M:\tau).G \mid t\langle M\rangle \]
\[ \mid \text{end} \]

\[ \tau, \sigma ::= \Pi x:\tau.\sigma \mid \Sigma x:\tau.\sigma \mid b \mid S(M) \]

Singleton Types

- \( S(M) \), where \( M : b \).
- \( S(M) \) denotes a value of type \( b \) equal to \( M \).
- \( 4 : \text{Nat}(4), \, x:\text{Nat}(4) \vdash 4 \equiv x : \text{Nat} \).
Certifying Data in MPST

Types

Global and Message Types

\[ G ::= p \rightarrow q : (x:\tau).G \]
\[ p \rightarrow q : (l_j:G_j)_{j \in J} \]
\[ p \rightarrow q : (T).G \]
\[ \mu t \ (x = M:\tau).G \mid t\langle M \rangle \]
\[ \text{end} \]

\[ \tau, \sigma ::= \Pi x:\tau.\sigma \mid \Sigma x:\tau.\sigma \mid b \mid S(M) \]

Singleton Types

- \(S(M)\), where \(M : b\).
- \(S(M)\) denotes a value of type \(b\) equal to \(M\).
- \(4 : \text{Nat}(4)\), \(x:\text{Nat}(4) \vdash 4 \equiv x : \text{Nat}\).
Certifying Data in MPST

Challenges

- Well-formedness of global types with dependencies:
  
  - Projection:
    - Ensure that endpoints know all contents of sent messages.
    - Generate message types that "make sense" for the recipients.
    - Preserve global dependencies on a local level.
Certifying Data in MPST

Challenges

- Well-formedness of global types with dependencies:
  - Global types often specify data dependencies between multiple endpoints:
    
    \[ p \rightarrow q : (x : \tau_1). \]

    \[ p \rightarrow s : (y : \mathcal{P}(x)).G \]
Certifying Data in MPST
Challenges

- Well-formedness of global types with dependencies:
  - Global types often specify data dependencies between multiple endpoints:
    \[ p \to q : (x : \tau_1).p \to s : (y : \mathcal{P}(x)).G \]
  - \( \mathcal{P}(x) \) is sent to s, but s may not know \( x \) (but p does!).
Certifying Data in MPST
Challenges

- Well-formedness of global types with dependencies:
  - Global types often specify data dependencies between multiple endpoints:
    \[ p \rightarrow q : (x : \tau_1).p \rightarrow s : (y : \mathcal{P}(x)).G \]
  - \( \mathcal{P}(x) \) is sent to \( s \), but \( s \) may not know \( x \) (but \( p \) does!).

- Projection:
  - Ensure that endpoints know all contents of sent messages.
  - Generate message types that “make sense” for the recipients.
Certifying Data in MPST
Challenges

- Well-formedness of global types with dependencies:
  - Global types often specify data dependencies between multiple endpoints:
    \[ p \rightarrow q : (x : \tau_1).p \rightarrow s : (y : P(x)).G \]
  
  - \( P(x) \) is sent to \( s \), but \( s \) may not know \( x \) (but \( p \) does!).

- Projection:
  - Ensure that endpoints know all contents of sent messages.
  - Generate message types that “make sense” for the recipients.
  - Preserve global dependencies on a local level.
Certifying Data in MPST

Design Choices

- Maintain a general specification language (type theory).
- Specs may not be decidable (need explicit proof!).

Bernardo Toninho, Nobuko Yoshida
Certifying and Signing Data in Multiparty Session Types
Certifying Data in MPST
Design Choices

- Maintain a general specification language (type theory).
- Specs may not be decidable (need explicit proof!).
- Global type well-formedness:
  - Projection of a well-formed global type produces well-formed local types by a simple projection rule;
  - If a collection of processes satisfying local types exist, the global specification is satisfied.
Consider the global type:

\[ G = p \rightarrow q : (x: \text{Nat}).p \rightarrow r : (y:x < 10).G' \]
Certifying Data in MPST
Projection – A naive attempt

Consider the global type:

\[ G = p \rightarrow q : (x : \text{Nat}). p \rightarrow r : (y : x < 10). G' \]

An attempt at \( G \upharpoonright p \) and \( G \upharpoonright r \):

\[ \]
Consider the global type:

\[ G = p \rightarrow q : (x : \text{Nat}).p \rightarrow r : (y : x < 10).G' \]

An attempt at \( G \upharpoonright p \) and \( G \upharpoonright r \):

\[ G \upharpoonright p \overset{?}{=} q!(x : \text{Nat}); r!(y : x < 10); (G' \upharpoonright p) \]
Certifying Data in MPST

Projection – A naive attempt

Consider the global type:

\[ G = p \to q : (x: \text{Nat}). p \to r : (y : x < 10). G' \]

An attempt at \( G \upharpoonright p \) and \( G \upharpoonright r \):

\[ G \upharpoonright p \overset{?}{=} q!(x: \text{Nat}); r!(y : x < 10); (G' \upharpoonright p) \]

\[ G \upharpoonright r \overset{?}{=} p?(y : x < 10); (G' \upharpoonright r) \]
Certifying Data in MPST

Projection – A naive attempt

Consider the global type:

\[ G = p \rightarrow q : (x: \text{Nat}). p \rightarrow r : (y:x < 10).G' \]

An attempt at \( G \upharpoonright p \) and \( G \upharpoonright r \):

\[ G \upharpoonright p \models q!(x: \text{Nat}); r!(y:x < 10); (G' \upharpoonright p) \]

\[ G \upharpoonright r \models p?(y:x < 10); (G' \upharpoonright r) \]

Participant r does not know \( x \)!
Local types must only refer to locally available msg. variables;

constraints/Dependencies must be preserved:

\[ G = p \rightarrow q : (x:\text{Nat}).p \rightarrow r : (y:x < 10).G' \]

\[ G \upharpoonright r = p?(y:x < 10); (G' \upharpoonright r) \]
Certifying Data in MPST
Projection - Data Dependencies

- Local types must only refer to locally available msg. variables;
- constraints/Dependencies must be preserved:

\[ G = p \rightarrow q : (x: \text{Nat}).p \rightarrow r : (y : x < 10).G' \]

\[ G \mid r = p?(y : \Sigma x : \text{Nat}.x < 10); (G' \mid r) \]
Certifying Data in MPST
Projection - Data Dependencies

- Local types must only refer to locally available msg. variables;
- constraints/Dependences must be preserved:

\[ G = p \rightarrow q : (x: \text{Nat}). p \rightarrow r : (y: x < 10). G' \]

\[ G \upharpoonright r = p?(y: \Sigma x: \text{Nat}. x < 10); (G' \upharpoonright r) \]

\[ G \upharpoonright p \overline{?} q!(x: \text{Nat}); r!(y: \Sigma x: \text{Nat}. x < 10); (G' \upharpoonright p) \]
Certifying Data in MPST
Projection - Data Dependencies

- Local types must only refer to locally available msg. variables;
- constraints/Dependencies must be preserved:

\[ G = p \to q : (x:\text{Nat}).p \to r : (y : x < 10).G' \]

\[ G \upharpoonright r = p?(y:\Sigma x:\text{Nat}.x < 10); (G' \upharpoonright r) \]

\[ G \upharpoonright p = q!(x:\text{Nat}); r!(y:\Sigma x' : \text{Nat}(x).x' < 10); (G' \upharpoonright p) \]

Use \( \Sigma \)-types to existentially quantify on the receiver side.
Use singletons to preserve dependencies on the sender side.
Certifying Data in MPST
Projection - Data Dependencies

- Local types must only refer to locally available msg. variables;
- constraints/Dependencies must be preserved:

\[ G = p \rightarrow q : (x: \text{Nat}). p \rightarrow r : (y:x < 10). G' \]

\[ G \upharpoonright r = p?(y:\Sigma x: \text{Nat}. x < 10); (G' \upharpoonright r) \]

\[ G \upharpoonright p = q!(x: \text{Nat}); r!(y:\Sigma x': \text{Nat}(x). x' < 10); (G' \upharpoonright p) \]

Proposed solution:
- Use \( \Sigma \)-types to existentially quantify on the receiver side.
- Use singletons to preserve dependencies on the sender side.
Certifying Data in MPST
Projection - Data Dependencies

\[ G \upharpoonright r = p? (y: \Sigma x: \text{Nat}. x < 10); (G' \upharpoonright r) \]
\[ G \upharpoonright p = q! (x: \text{Nat}); r! (y: \Sigma x': \text{Nat}(x). x' < 10); (G' \upharpoonright p) \]
Certifying Data in MPST
Projection - Data Dependencies

\[ G \upharpoonright r = p?(y: \Sigma x: \text{Nat}. x < 10); (G' \upharpoonright r) \]

\[ G \upharpoonright p = q!(x: \text{Nat}); r!(y: \Sigma x': \text{Nat}(x). x' < 10); (G' \upharpoonright p) \]

How do we match the input and output types?
Certifying Data in MPST
Projection - Data Dependencies

\[ G \upharpoonright r = p? (y:\Sigma x:\text{Nat}. x < 10); (G' \upharpoonright r) \]

\[ G \upharpoonright p = q!(x:\text{Nat}); r!(y:\Sigma x':\text{Nat}(x). x' < 10); (G' \upharpoonright p) \]

How do we match the input and output types? Subtyping!
Certifying Data in MPST
Projection - Data Dependencies

\[G \upharpoonright r = p?(y:\Sigma x: \text{Nat}. x < 10); (G' \upharpoonright r)\]
\[G \upharpoonright p = q!(x: \text{Nat}); r!(y: \Sigma x': \text{Nat}(x). x' < 10); (G' \upharpoonright p)\]

How do we match the input and output types? Subtyping!

\[\Psi \vdash M : b\]
\[\Psi \vdash S(M) \leq b\]
Certifying Data in MPST
Projection - Data Dependencies

\[ G \upharpoonright r = p? (y: \Sigma x: \text{Nat}. x < 10); (G' \upharpoonright r) \]
\[ G \upharpoonright p = q!(x: \text{Nat}); r!(y: \Sigma x': \text{Nat}(x). x' < 10); (G' \upharpoonright p) \]

How do we match the input and output types? Subtyping!

\[ \Psi \vdash M : b \quad \Psi \vdash \tau \leq \tau' \quad \Psi, x: \tau' \vdash T \leq T' \]
\[ \Psi \vdash S(M) \leq b \quad \Psi \vdash p?(x: \tau). T \leq p?(x: \tau'). T' \]

\[ \Psi \vdash \tau' \leq \tau \quad \Psi, x: \tau \vdash T \leq T' \]
\[ \Psi \vdash p!(x: \tau). T \leq p!(x: \tau'). T' \]
We use a function $CTB$ to generate compatible type bindings for the sender.
Certifying Data in MPST

Projection

- We use a function $CTB$ to generate compatible type bindings for the sender.
- Use a singleton erasure function † for the recipient.
Certifying Data in MPST

Projection

- We use a function $CTB$ to generate compatible type bindings for the sender.
- Use a singleton erasure function $†$ for the recipient.
Certifying Data in MPST

Projection

- We use a function $CTB$ to generate compatible type bindings for the sender.
- Use a singleton erasure function $\dagger$ for the recipient.

Projection – Data I/O

\[
s \to r : (x : \tau).G' \ | \ p = \begin{cases} 
\end{cases}
\]
We use a function $CTB$ to generate compatible type bindings for the sender.

Use a singleton erasure function $\dagger$ for the recipient.

\[
s \rightarrow r : (x : \tau).G' \upharpoonright p = \begin{cases} 
  r!(x : (CTB(x : \tau)); (G' \upharpoonright p) & \text{if } p = s 
\end{cases}
\]
We use a function $CTB$ to generate compatible type bindings for the sender.

Use a singleton erasure function $\dagger$ for the recipient.

**Projection – Data I/O**

\[
\begin{align*}
 s \rightarrow r : (x : \tau). G' \upharpoonright p &= \begin{cases} 
 r!(x:(CTB(x:\tau))); (G' \upharpoonright p) & \text{if } p = s \\
 s?(x:(CTB(x:\tau))\dagger); (G' \upharpoonright p) & \text{if } p = r \\
 G' \upharpoonright p & \text{otherwise}
\end{cases}
\end{align*}
\]
Recall our initial example:

\[
G_{BM} = B \rightarrow \text{AirMalta} : (d_1:\text{String}, d_2:\text{Int}).
\]

\[
\text{AirMalta} \rightarrow B : (f:\text{String}).
\]

\[
B \rightarrow \text{Booking} : (\text{String}(d_1), \text{Int}(d_2)).
\]

\[
\text{Booking} \rightarrow B : (\text{opt} : \text{List} \text{String}).
\]

\[
B \rightarrow \text{Booking} : (\Sigma s : \text{String}. s \in \text{opt}).
\]

\[
\text{Booking} \rightarrow B : (r : \text{Int}).
\]

\[
B \rightarrow \text{COST} : (\text{String}(f), \text{Int}(r)).
\]

\[
\text{COST} \rightarrow B : (\text{ok} : \textbf{end}, \text{nok} : \textbf{end})
\]
Certifying Data in MPST

Projection

Recall our initial example:

\[ G_{BM} = B \rightarrow \text{AirMalta} : (d_1: \text{String}, d_2: \text{Int}). \]

AirMalta \rightarrow B : (f: \text{String}).

B \rightarrow \text{Booking} : (\text{String}(d_1), \text{Int}(d_2)).

Booking \rightarrow B : (opt: \text{List String}).

B \rightarrow \text{Booking} : (\sum s: \text{String}. s \in opt).

Booking \rightarrow B : (r: \text{Int}).

B \rightarrow \text{COST} : (\text{String}(f), \text{Int}(r)).

\text{COST} \rightarrow B : (\text{ok} : \text{end}, \text{nok} : \text{end})

\[ G_{BM} \upharpoonright B = \text{AirMalta!}(d_1: \text{String}, d_2: \text{Int}).\text{AirMalta?}(f: \text{String}). \]
Recall our initial example:

\[
G_{BM} = \quad B \rightarrow \text{AirMalta} : (d_1: \text{String}, d_2: \text{Int}). \\
\text{AirMalta} \rightarrow B : (f: \text{String}). \\
B \rightarrow \text{Booking} : (\text{String}(d_1), \text{Int}(d_2)). \\
\text{Booking} \rightarrow B : (\text{opt}: \text{List String}). \\
B \rightarrow \text{Booking} : (\Sigma s: \text{String}. s \in \text{opt}). \\
\text{Booking} \rightarrow B : (r: \text{Int}). \\
B \rightarrow \text{COST} : (\text{String}(f), \text{Int}(r)). \\
\text{COST} \rightarrow B : (\text{ok} : \text{end}, \text{nok} : \text{end})
\]

\[
G_{BM} \upharpoonright B = \quad \text{AirMalta}! (d_1: \text{String}, d_2: \text{Int}). \text{AirMalta}? (f: \text{String}). \\
\text{Book}! (\Sigma d'_1: \text{String}(d_1), d'_2: \text{Int}(d_2). \text{String}(d'_1), \text{Int}(d'_2)).
\]
Certifying Data in MPST
Projection

Recall our initial example:

\[
G_{BM} = B \rightarrow \text{AirMalta} : (d_1:\text{String}, d_2:\text{Int}).
\]

\[
\text{AirMalta} \rightarrow B : (f:\text{String}).
\]

\[
B \rightarrow \text{Booking} : (\text{String}(d_1), \text{Int}(d_2)).
\]

\[
\text{Booking} \rightarrow B : (\text{opt}:\text{List String}).
\]

\[
B \rightarrow \text{Booking} : (\sum s:\text{String}. s \in \text{opt}).
\]

\[
\text{Booking} \rightarrow B : (r:\text{Int}).
\]

\[
B \rightarrow \text{COST} : (\text{String}(f), \text{Int}(r)).
\]

\[
\text{COST} \rightarrow B : (\text{ok} : \text{end}, \text{nok} : \text{end})
\]

\[
G_{BM} \upharpoonright B = \text{AirMalta!}(d_1:\text{String}, d_2:\text{Int}).\text{AirMalta?}(f:\text{String}).
\]

\[
\text{Book!}(\sum d'_1:\text{String}(d_1), d'_2:\text{Int}(d_2).\text{String}(d'_1), \text{Int}(d'_2)).
\]

\[
\text{Book?}(\text{opt}:\text{List String}).\text{Booking!}(\sum s:\text{String}. s \in \text{opt}).
\]
Certifying Data in MPST

Projection

Recall our initial example:

\[ G_{BM} = B \rightarrow \text{AirMalta} : (d_1:\text{String}, d_2:\text{Int}). \]
\[ \text{AirMalta} \rightarrow B : (f:\text{String}). \]
\[ B \rightarrow \text{Booking} : (\text{String}(d_1), \text{Int}(d_2)). \]
\[ \text{Booking} \rightarrow B : (opt:\text{List String}). \]
\[ B \rightarrow \text{Booking} : (\sum s:\text{String}. s \in opt). \]
\[ \text{Booking} \rightarrow B : (r:\text{Int}). \]
\[ B \rightarrow \text{COST} : (\text{String}(f), \text{Int}(r)). \]
\[ \text{COST} \rightarrow B : (\text{ok} : \text{end}, \text{nok} : \text{end}) \]

\[ G_{BM} \vdash B = \text{AirMalta}!(d_1:\text{String}, d_2:\text{Int}). \text{AirMalta}? (f:\text{String}). \]
\[ \text{Book}! (\sum d'_1:\text{String}(d_1), d'_2:\text{Int}(d_2). \text{String}(d'_1), \text{Int}(d'_2)). \]
\[ \text{Book}? (opt:\text{List String}). \text{Booking}! (\sum s:\text{String}. s \in opt). \]
\[ \text{Book}? (r:\text{Int}). \text{COST}! (\sum f':\text{String}(f), r':\text{Int}(r). \text{String}(f'), \text{Int}(r')). \]
...
Certifying Data in MPST
Projection – Methodology

To summarize:

▶ Given a well-formed global type \( G \)
▶ Project well-formed local types \( \{ G \upharpoonright p_i \}_{i \in \text{pid}(G)} \), preserving dependencies.
▶ Find endpoint processes that satisfy each of \( G \upharpoonright p_i \).
Results
Type Safety

Typing Judgment

$\Psi; \Gamma; \Delta \vdash P$  $\Psi \vdash M : \tau$

Process $P$ uses its session channels as specified in $\Delta$; shared channels as specified in $\Gamma$ and message variables as in $\Psi$. 
Results
Type Safety

Typing Judgment

\[ \Psi; \Gamma; \Delta \vdash P \quad \Psi \vdash M : \tau \]

Process \( P \) uses its session channels as specified in \( \Delta \); shared channels as specified in \( \Gamma \) and message variables as in \( \Psi \).

Substitution

If \( \Psi, x: \tau, \Psi'; \Gamma; \Delta \vdash P \) and \( \Psi \vdash M : \tau \) then
\[ \Psi, \Psi' \{M/x\}; \Gamma; \Delta\{M/x\} \vdash P\{M/x\} . \]
Results
Type Safety

Typing Judgment

\[ \Psi; \Gamma; \Delta \vdash P \quad \Psi \vdash M : \tau \]

Process \( P \) uses its session channels as specified in \( \Delta \); shared channels as specified in \( \Gamma \) and message variables as in \( \Psi \).

Subject Transition

If \( \Psi; \Gamma; \Delta \vdash P \) and \( P \xrightarrow{\alpha} P' \) then:
Results
Type Safety

Typing Judgment

$$\Psi; \Gamma; \Delta \vdash P \quad \Psi \vdash M : \tau$$

Process $P$ uses its session channels as specified in $\Delta$; shared channels as specified in $\Gamma$ and message variables as in $\Psi$.

Subject Transition

If $\Psi; \Gamma; \Delta \vdash P$ and $P \xrightarrow{\alpha} P'$ then:
Results
Type Safety

Typing Judgment

\[ \Psi; \Gamma; \Delta \vdash P \quad \Phi \vdash M : \tau \]

Process \( P \) uses its session channels as specified in \( \Delta \); shared channels as specified in \( \Gamma \) and message variables as in \( \Psi \).

Subject Transition

If \( \Psi; \Gamma; \Delta \vdash P \) and \( P \xrightarrow{\alpha} P' \) then:

(a) If \( \alpha \) is an output, selection or action on a shared name, there exists \( \Delta' \) s.t. \( \Psi; \Gamma; \Delta' \vdash P' \) and \( \langle \Gamma; \Delta \rangle \xrightarrow{\alpha} \langle \Gamma; \Delta' \rangle \).
Results
Type Safety

Typing Judgment

\[ \Psi; \Gamma; \Delta \vdash P \quad \Psi \vdash M : \tau \]

Process \( P \) uses its session channels as specified in \( \Delta \); shared channels as specified in \( \Gamma \) and message variables as in \( \Psi \).

Subject Transition

If \( \Psi; \Gamma; \Delta \vdash P \) and \( P \xrightarrow{\alpha} P' \) then:

(a) If \( \alpha \) is an output, selection or action on a shared name, there exists \( \Delta' \) s.t \( \Psi; \Gamma; \Delta' \vdash P' \) and \( \langle \Gamma; \Delta \rangle \xrightarrow{\alpha} \langle \Gamma; \Delta' \rangle \).

(b) If \( \alpha \) is an input or a branching then for all \( \Delta' \), if \( \langle \Gamma; \Delta \rangle \xrightarrow{\alpha} \langle \Gamma; \Delta' \rangle \) then \( \Psi; \Gamma; \Delta' \vdash P' \).
Applications
Digital Certificates

- Properties expressed via dependencies require explicit proof.
Applications
Digital Certificates

- Properties expressed via dependencies require explicit proof.
- In general, might be too strong of a requirement:
  - Sometimes we (partially) trust communicating parties.
Applications
Digital Certificates

- Properties expressed via dependencies require explicit proof.
- In general, might be too strong of a requirement:
  - Sometimes we (partially) trust communicating parties.
  - Exchange digital proof certificates instead?
Applications
Digital Certificates

▶ Properties expressed via dependencies require explicit proof.
▶ In general, might be too strong of a requirement:
  ▶ Sometimes we (partially) trust communicating parties.
  ▶ Exchange digital proof certificates instead?
▶ Combine signed messages $\diamond_p \tau$ and erasable $[\tau]$.
Applications
Digital Certificates

- Properties expressed via dependencies require explicit proof.
- In general, might be too strong of a requirement:
  - Sometimes we (partially) trust communicating parties.
  - Exchange digital proof certificates instead?
- Combine signed messages $\diamond_p \tau$ and erasable $[\tau]$.
- A message $M : \diamond_p [\tau]$ denotes:
  - A term $M$ signed by participant $p$ (public key infrastructure).
  - Denotes the existence of a proof of $\tau$.
  - May not itself contain such a proof (erasure of $[]$’d terms).
Applications
Signed Messages

▶ \( \Diamond_p \tau \) is a strong (role-indexed) monad.
Applications
Signed Messages

- $\diamond_p \tau$ is a strong (role-indexed) monad.
- “Diamonds are forever” – From $M : \diamond_p \tau$ cannot extract $N : \tau$. 
Applications
Signed Messages

- $\Diamond_p \tau$ is a strong (role-indexed) monad.
- "Diamonds are forever" – From $M : \Diamond_p \tau$ cannot extract $N : \tau$.
- A process can sign a message with any role.
- ... but global types help with this.
Applications
Signed Messages

- \(\Diamond_p \tau\) is a strong (role-indexed) monad.
- “Diamonds are forever” – From \(M : \Diamond_p \tau\) cannot extract \(N : \tau\).
- A process can sign a message with any role.
- ... but global types help with this.

Well-signed Global Type (informally)

If \(p \to q : (\Diamond_{p'} \tau).G' \sqsubseteq G\) then \(p' = p\) or \(r \to p : (\Diamond_{p'} \tau).G'' \sqsubseteq G\)

“earlier”.
Applications
Signed Messages

- $\diamond_p \tau$ is a strong (role-indexed) monad.
- “Diamonds are forever” – From $M : \diamond_p \tau$ cannot extract $N : \tau$.
- A process can sign a message with any role.
- ... but global types help with this.

Well-signed Global Type (informally)
Signatures in $G$ cannot be forged.

- Signatures allow us to encode provenance information.
Applications
Digital Certificates

\[ G_{BM} = B \rightarrow \text{AirMalta} : (d_1:\text{String}, d_2:\text{Int}). \]

\[ \text{AirMalta} \rightarrow B : (f:\text{String}). \]

\[ B \rightarrow \text{Booking} : (\text{String}(d_1), \text{Int}(d_2)). \]

\[ \text{Booking} \rightarrow B : (\text{opt}:\text{List String}). \]

\[ B \rightarrow \text{Booking} : (\Sigma s:\text{String}. s \in \text{opt}). \]

\[ \text{Booking} \rightarrow B : (r:\text{Int}). \]

\[ B \rightarrow \text{COST} : (\text{String}, \text{Int}). \]

\[ \text{COST} \rightarrow B : (\text{ok} : \text{end}; \]

\[ \text{nok} : \text{end}) \]
Applications
Digital Certificates

\[ G_{BM} = \]
B → AirMalta : \((d_1: \text{String}, d_2: \text{Int})\).
AirMalta → B : \((f: \diamond_{AM} \text{String})\).
B → Booking : \((\text{String}(d_1), \text{Int}(d_2))\).
Booking → B : \((\text{opt: List String})\).
B → Booking : \((\Sigma s: \text{String}. s \in \text{opt})\).
Booking → B : \((r: \diamond_{BK} \text{Int})\).
B → COST : \((\diamond_{AM} \text{String}, \diamond_{BK} \text{Int})\).
COST → B : \((\text{ok : end};\)
\n    \text{nok : end})

Trust no one, sign “everything”.

Bernardo Toninho, Nobuko Yoshida
Certifying and Signing Data in Multiparty Session Types
Applications
Digital Certificates

\[
G_{BM} = \begin{align*}
B & \rightarrow \text{AirMalta} : (d_1: \text{String}, d_2: \text{Int}). & \text{Dest. + date} \\
\text{AirMalta} & \rightarrow B : (f: \text{String}, \diamond_{AM}[\text{String}]). & \text{Flight Info.} \\
B & \rightarrow \text{Booking} : (\text{String}(d_1), \text{Int}(d_2)). & \text{Dest. + date} \\
\text{Booking} & \rightarrow B : (\text{opt}: \text{List String}). & \text{Options} \\
B & \rightarrow \text{Booking} : (\sum s: \text{String}. s \in \text{opt}). & \text{Choose} \\
\text{Booking} & \rightarrow B : (r: \text{Int}, \diamond_{BK}[\text{Int}]). & \text{Receipt} \\
B & \rightarrow \text{COST} : (\diamond_{AM}[\text{String}], \diamond_{BK}[\text{Int}]). & \text{Expense claims} \\
\text{COST} & \rightarrow B : (\text{ok} : \text{end}; \\n& \text{nok} : \text{end}) & \text{ok or not}
\end{align*}
\]

Trust a bit more, allow for erasure at runtime.
Applications
Digital Certificates

A slight variation:

\[ G_{BM} = \text{Booking} \rightarrow B : (\text{opt}:\text{List String}). \]
\[ \text{Options} \]
\[ B \rightarrow \text{Booking} : (c:\Sigma s:\text{String}.s \in \text{opt}). \]
\[ \text{Choose} \]
\[ \text{Booking} \rightarrow B : (r:\Sigma i:\text{Int}.\text{receipt}(\pi_1(c))). \]
\[ \text{Better Receipt} \]
\[ B \rightarrow \text{COST} : (\text{Int}(\pi_1(r))). \]
\[ \text{Claim} \]
\[ \ldots \]
Applications
Digital Certificates

A slight variation:

\[ G_{BM} = \text{Booking} \rightarrow B : (opt : \text{List String}). \]
\[ B \rightarrow \text{Booking} : (c : \Sigma s : \text{String}. s \in opt). \]
\[ \text{Booking} \rightarrow B : (r : \Sigma i : \text{Int}. \text{receipt}(\pi_1(c))). \]
\[ B \rightarrow \text{COST} : (\text{Int}(\pi_1(r))). \]

\[ G_{BM} \upharpoonright \text{COST} = B? (\Sigma opt : \ldots , c : \ldots , r : (\Sigma i : \text{Int}. \text{receipt}(\pi_1(c))). \text{Int}(\pi_1(r))) \]
Applications
Digital Certificates

A slight variation:

\[ G_{BM} = \text{Booking} \to B : (\text{opt} : \text{List String}). \]
\[ B \to \text{Booking} : (c : \Sigma s : \text{String}. s \in \text{opt}). \]
\[ \text{Booking} \to B : (r : \Sigma i : \text{Int}. [\text{receipt}(\pi_1(c))] ). \]
\[ B \to \text{COST} : (\text{Int}(\pi_1(r))). \]

Options
Choose
Better Receipt
Claim

...
Applications
Digital Certificates

A slight variation:

\[ G_{BM} = \text{Booking} \to B : (opt: \text{List String}). \]
\[ B \to \text{Booking} : (c: \Sigma s: \text{String}. s \in opt). \]
\[ \text{Booking} \to B : (r: \Sigma i: \text{Int}. \text{receipt}(\pi_1(c))). \]
\[ B \to \text{COST} : (\text{Int}(\pi_1(r))). \]

... 

Using the runtime-erased global type \( G_{BM}^{\downarrow} \):

\[ G_{BM}^{\downarrow} \uparrow \text{COST} = B?(\text{Int}) \]

Erasure gives us both “trust” and (potential) optimizations!
Concluding Remarks

- We have introduced a framework for certifying data in MPST.
- Properties in global types consistently mapped to local types.
- Explicit proof object exchange as witnesses to properties.
- Digital proof certificates introduce considerations of trust.
- Future work:
  - Dynamic monitoring.
  - Auto. proof generation.
  - Implementation