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.

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.
- 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

$$
\begin{array}{lcll}
G_{BM} & = & \mathsf{B} \to \mathsf{AirMalta} : (\mathsf{String}, \mathsf{Int}). & \text{Destination + date} \\
 & & \mathsf{AirMalta} \to \mathsf{B} : (\mathsf{String}). & \text{Flight Info.}
\end{array}
$$

Introduction

Trip to Malta

$$
\begin{array}{lcll}
G_{BM} & = & \mathsf{B} \to \mathsf{AirMalta} : (\mathsf{String}, \mathsf{Int}). & \text{Destination} + \text{date} \\
 & & \mathsf{AirMalta} \to \mathsf{B} : (\mathsf{String}). & \text{Flight Info.} \\
 & & \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}, \mathsf{Int}). & \text{Destination} + \text{date}
\end{array}
$$

Introduction

Trip to Malta

$$
\begin{array}{lll}
G_{BM} = & \mathsf{B} \to \mathsf{AirMalta} : (\mathsf{String}, \mathsf{Int}). & \text{Destination + date} \\
& \mathsf{AirMalta} \to \mathsf{B} : (\mathsf{String}). & \text{Flight Info.} \\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}, \mathsf{Int}). & \text{Destination + date} \\
& \mathsf{Booking} \to \mathsf{B} : (\mathsf{List\ String}). & \text{Options} \\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}). & \text{Choose} \\
& \mathsf{Booking} \to \mathsf{B} : (\mathsf{Int}). & \text{Receipt}
\end{array}
$$

Introduction

Trip to Malta

$$
\begin{array}{lll}
G_{BM} = & \mathsf{B} \to \mathsf{AirMalta} : (\mathsf{String}, \mathsf{Int}). & \text{Destination + date} \\
& \mathsf{AirMalta} \to \mathsf{B} : (\mathsf{String}). & \text{Flight Info.} \\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}, \mathsf{Int}). & \text{Destination + date} \\
& \mathsf{Booking} \to \mathsf{B} : (\mathsf{List\ String}). & \text{Options} \\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}). & \text{Choose} \\
& \mathsf{Booking} \to \mathsf{B} : (\mathsf{Int}). & \text{Receipt} \\
& \mathsf{B} \to \mathsf{COST} : (\mathsf{String}, \mathsf{Int}). & \text{Expense claims} \\
& \mathsf{ICClaims} \to \mathsf{B} : (\mathsf{ok} : \mathbf{end}; & \text{ok or not} \\
& \qquad\qquad\qquad\quad \mathsf{nok} : \mathbf{end}) &
\end{array}
$$

Introduction

Trip to Malta

$$
\begin{array}{lll}
G_{BM} = & \mathsf{B} \to \mathsf{AirMalta} : (\mathsf{String}, \mathsf{Int}). & \text{Destination + date} \\
& \mathsf{AirMalta} \to \mathsf{B} : (\mathsf{String}). & \text{Flight Info.} \\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}, \mathsf{Int}). & \text{Destination + date} \\
& \mathsf{Booking} \to \mathsf{B} : (\mathsf{List\ String}). & \text{Options} \\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}). & \text{Choose} \\
& \mathsf{Booking} \to \mathsf{B} : (\mathsf{Int}). & \text{Receipt} \\
& \mathsf{B} \to \mathsf{COST} : (\mathsf{String}, \mathsf{Int}). & \text{Expense claims} \\
& \mathsf{ICClaims} \to \mathsf{B} : (\mathsf{ok} : \mathbf{end}; & \text{ok or not} \\
& \qquad\qquad\qquad\qquad \mathsf{nok} : \mathbf{end}) &
\end{array}
$$

- 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

$$
\begin{array}{lcll}
G_{BM} & = & \mathsf{B} \to \mathsf{AirMalta} : (d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}). & \text{Dest. } + \text{ date} \\
 & & \mathsf{AirMalta} \to \mathsf{B} : (f{:}\mathsf{String}). & \text{Flight Info.} \\
 & & \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}(d_1), \mathsf{Int}(d_2)). & \text{Dest. } + \text{ date}
\end{array}
$$

Certifying Data in MPST

Revisiting the trip to Malta

$$
\begin{array}{lcll}
G_{BM} & = & \mathsf{B} \to \mathsf{AirMalta} : (d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}). & \text{Dest. + date} \\
 & & \mathsf{AirMalta} \to \mathsf{B} : (f{:}\mathsf{String}). & \text{Flight Info.} \\
 & & \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}(d_1), \mathsf{Int}(d_2)). & \text{Dest. + date} \\
 & & \mathsf{Booking} \to \mathsf{B} : (opt{:}\mathsf{List\ String}). & \text{Options} \\
 & & \mathsf{B} \to \mathsf{Booking} : (\Sigma s{:}\mathsf{String}.s \in opt). & \text{Choose} \\
 & & \mathsf{Booking} \to \mathsf{B} : (r{:}\mathsf{Int}). & \text{Receipt}
\end{array}
$$

Certifying Data in MPST

Revisiting the trip to Malta

$$
\begin{array}{lll}
G_{BM} = & \mathsf{B} \to \mathsf{AirMalta} : (d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}). & \text{Dest. + date} \\
& \mathsf{AirMalta} \to \mathsf{B} : (f{:}\mathsf{String}). & \text{Flight Info.} \\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}(d_1), \mathsf{Int}(d_2)). & \text{Dest. + date} \\
& \mathsf{Booking} \to \mathsf{B} : (opt{:}\mathsf{List\ String}). & \text{Options} \\
& \mathsf{B} \to \mathsf{Booking} : (\Sigma s{:}\mathsf{String}.s \in opt). & \text{Choose} \\
& \mathsf{Booking} \to \mathsf{B} : (r{:}\mathsf{Int}). & \text{Receipt} \\
& \mathsf{B} \to \mathsf{COST} : (\mathsf{String}(f), \mathsf{Int}(r)). & \text{Expense claims} \\
& \mathsf{ICClaims} \to \mathsf{B} : (\mathsf{ok} : \mathbf{end}; & \text{ok or not} \\
& \qquad\qquad \mathsf{nok} : \mathbf{end}) &
\end{array}
$$

- Type families and dependent tuples allow for a richer spec.
- Singleton types (e.g. $\mathsf{String}(d)$) specify equality constraints.

Certifying Data in MPST

Types

Global and Message Types

$$
\begin{array}{lcl}
G & ::= & \mathsf{p} \to \mathsf{q} : (x{:}\tau).G \\
 & \mid & \mathsf{p} \to \mathsf{q} : \left(\mathsf{l}_j{:}G_j\right)_{j \in J} \\
 & \mid & \mathsf{p} \to \mathsf{q} : (T).G \\
 & \mid & \mu t\,(x = M{:}\tau).G \mid t\langle M \rangle \\
 & \mid & \mathbf{end} \\
 & & \\
\tau, \sigma & ::= &
\end{array}
$$

Certifying Data in MPST

Types

Global and Message Types

$$
\begin{array}{lcl}
G & ::= & \mathsf{p} \to \mathsf{q} : (x{:}\tau).G \\
 & \mid & \mathsf{p} \to \mathsf{q} : \left(\mathsf{l}_j{:}G_j\right)_{j \in J} \\
 & \mid & \mathsf{p} \to \mathsf{q} : (T).G \\
 & \mid & \mu t\,(x = M{:}\tau).G \mid t\langle M \rangle \\
 & \mid & \mathbf{end} \\
 & & \\
\tau, \sigma & ::= & \Pi x{:}\tau.\sigma
\end{array}
$$

Certifying Data in MPST

Types

Global and Message Types

$$
\begin{array}{lcl}
G & ::= & \mathsf{p} \to \mathsf{q} : (x{:}\tau).G \\
 & \mid & \mathsf{p} \to \mathsf{q} : \left(\mathsf{l}_j{:}G_j\right)_{j \in J} \\
 & \mid & \mathsf{p} \to \mathsf{q} : (T).G \\
 & \mid & \mu t\,(x = M{:}\tau).G \mid t\langle M \rangle \\
 & \mid & \mathbf{end} \\
 & & \\
\tau, \sigma & ::= & \Pi x{:}\tau.\sigma \mid \Sigma x{:}\tau.\sigma \mid b
\end{array}
$$

Certifying Data in MPST

Types

Global and Message Types

$$
\begin{array}{lcl}
G & ::= & \mathsf{p} \to \mathsf{q} : (x{:}\tau).G \\
 & \mid & \mathsf{p} \to \mathsf{q} : \left(\mathsf{l}_j{:}G_j\right)_{j \in J} \\
 & \mid & \mathsf{p} \to \mathsf{q} : (T).G \\
 & \mid & \mu t\,(x = M{:}\tau).G \mid t\langle M \rangle \\
 & \mid & \mathbf{end} \\
 & & \\
\tau, \sigma & ::= & \Pi x{:}\tau.\sigma \mid \Sigma x{:}\tau.\sigma \mid b \mid \mathcal{S}(M)
\end{array}
$$

Certifying Data in MPST

Types

Global and Message Types

$$
\begin{array}{rcl}
G & ::= & \mathsf{p} \to \mathsf{q} : (x{:}\tau).G \\
& \mid & \mathsf{p} \to \mathsf{q} : \left(\mathsf{l}_j{:}G_j\right)_{j \in J} \\
& \mid & \mathsf{p} \to \mathsf{q} : (T).G \\
& \mid & \mu t\,(x = M{:}\tau).G \mid t\langle M \rangle \\
& \mid & \mathbf{end} \\
\\
\tau, \sigma & ::= & \Pi x{:}\tau.\sigma \mid \Sigma x{:}\tau.\sigma \mid b \mid \mathcal{S}(M)
\end{array}
$$

Singleton Types

- $\mathcal{S}(M)$, where $M : b$.
- $\mathcal{S}(M)$ denotes a value of type b equal to M.

Certifying Data in MPST

Types

Global and Message Types

$$
\begin{array}{rcl}
G & ::= & \mathsf{p} \to \mathsf{q} : (x{:}\tau).G \\
& \mid & \mathsf{p} \to \mathsf{q} : \left(\mathsf{l}_j{:}G_j\right)_{j \in J} \\
& \mid & \mathsf{p} \to \mathsf{q} : (T).G \\
& \mid & \mu t\,(x = M{:}\tau).G \mid t\langle M \rangle \\
& \mid & \mathbf{end} \\
\\
\tau, \sigma & ::= & \Pi x{:}\tau.\sigma \mid \Sigma x{:}\tau.\sigma \mid b \mid \mathcal{S}(M)
\end{array}
$$

Singleton Types

- $\mathcal{S}(M)$, where $M : b$.
- $\mathcal{S}(M)$ denotes a value of type b equal to M.
- $4 : \mathsf{Nat}(4)$, $x{:}\mathsf{Nat}(4) \vdash 4 \equiv x : \mathsf{Nat}$.

Certifying Data in MPST

Types

Global and Message Types

$$
\begin{array}{rcl}
G & ::= & \mathsf{p} \to \mathsf{q} : (x{:}\tau).G \\
& \mid & \mathsf{p} \to \mathsf{q} : \left(\mathsf{l}_j{:}G_j\right)_{j \in J} \\
& \mid & \mathsf{p} \to \mathsf{q} : (T).G \\
& \mid & \mu t\,(x = M{:}\tau).G \mid t\langle M \rangle \\
& \mid & \mathbf{end} \\
\\
\tau, \sigma & ::= & \Pi x{:}\tau.\sigma \mid \Sigma x{:}\tau.\sigma \mid b \mid \mathcal{S}(M)
\end{array}
$$

Singleton Types

- $\mathcal{S}(M)$, where $M : b$.
- $\mathcal{S}(M)$ denotes a value of type b equal to M.
- $4 : \mathrm{Nat}(4)$, $x{:}\mathrm{Nat}(4) \vdash 4 \equiv x : \mathrm{Nat}$.

Certifying Data in MPST

Challenges

- Well-formedness of global types with dependencies:

Certifying Data in MPST

Challenges

- Well-formedness of global types with dependencies:
  - Global types often specify data dependencies between multiple endpoints:

$$\mathsf{p} \to \mathsf{q} : (x : \tau_1).\mathsf{p} \to \mathsf{s} : (y : \mathcal{P}(x)).G$$

Certifying Data in MPST

Challenges

- Well-formedness of global types with dependencies:
  - Global types often specify data dependencies between multiple endpoints:

$$\mathsf{p} \to \mathsf{q} : (x : \tau_1).\mathsf{p} \to \mathsf{s} : (y : \mathcal{P}(x)).G$$

  - $\mathcal{P}(x)$ 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:

$$\mathsf{p} \to \mathsf{q} : (x : \tau_1).\mathsf{p} \to \mathsf{s} : (y : \mathcal{P}(x)).G$$

  - $\mathcal{P}(x)$ 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:

$$\mathsf{p} \to \mathsf{q} : (x : \tau_1).\mathsf{p} \to \mathsf{s} : (y : \mathcal{P}(x)).G$$

  - $\mathcal{P}(x)$ 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.
  - 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!).

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.

Certifying Data in MPST

Projection – A naive attempt

Consider the global type:

$$G = \mathsf{p} \to \mathsf{q} : (x{:}\mathsf{Nat}).\mathsf{p} \to \mathsf{r} : (y{:}x < 10).G'$$

Certifying Data in MPST

Projection – A naive attempt

Consider the global type:

$$G = \mathsf{p} \to \mathsf{q} : (x{:}\mathsf{Nat}).\mathsf{p} \to \mathsf{r} : (y{:}x < 10).G'$$

An attempt at $G \upharpoonright \mathsf{p}$ and $G \upharpoonright \mathsf{r}$:

Certifying Data in MPST

Projection – A naive attempt

Consider the global type:

$$G = \mathsf{p} \to \mathsf{q} : (x{:}\mathsf{Nat}).\mathsf{p} \to \mathsf{r} : (y{:}x < 10).G'$$

An attempt at $G \upharpoonright \mathsf{p}$ and $G \upharpoonright \mathsf{r}$:

$$G \upharpoonright \mathsf{p} \stackrel{?}{=} \mathsf{q}!(x{:}\mathsf{Nat}); \mathsf{r}!(y{:}x < 10); (G' \upharpoonright \mathsf{p})$$

Certifying Data in MPST

Projection – A naive attempt

Consider the global type:

$$G = \mathsf{p} \to \mathsf{q} : (x{:}\mathsf{Nat}).\mathsf{p} \to \mathsf{r} : (y{:}x < 10).G'$$

An attempt at $G \upharpoonright \mathsf{p}$ and $G \upharpoonright \mathsf{r}$:

$$G \upharpoonright \mathsf{p} \stackrel{?}{=} \mathsf{q}!(x{:}\mathsf{Nat}); \mathsf{r}!(y{:}x < 10); (G' \upharpoonright \mathsf{p})$$

$$G \upharpoonright \mathsf{r} \stackrel{?}{=} \mathsf{p}?(y{:}x < 10); (G' \upharpoonright \mathsf{r})$$

Certifying Data in MPST

Projection – A naive attempt

Consider the global type:

$$G = \mathsf{p} \to \mathsf{q} : (x{:}\mathsf{Nat}).\mathsf{p} \to \mathsf{r} : (y{:}x < 10).G'$$

An attempt at $G \upharpoonright \mathsf{p}$ and $G \upharpoonright \mathsf{r}$:

$$G \upharpoonright \mathsf{p} \stackrel{?}{=} \mathsf{q}!(x{:}\mathsf{Nat}); \mathsf{r}!(y{:}x < 10); (G' \upharpoonright \mathsf{p})$$

$$G \upharpoonright \mathsf{r} \stackrel{?}{=} \mathsf{p}?(y{:}x < 10); (G' \upharpoonright \mathsf{r})$$

Participant r does not know x!

Certifying Data in MPST

Projection - Data Dependencies

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

$$G = \mathsf{p} \to \mathsf{q} : (x{:}\mathsf{Nat}).\mathsf{p} \to \mathsf{r} : (y{:}x < 10).G'$$

$$G \upharpoonright \mathsf{r} = \mathsf{p}?(y{:}x < 10); (G' \upharpoonright \mathsf{r})$$

Certifying Data in MPST

Projection - Data Dependencies

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

$$G = \mathsf{p} \to \mathsf{q} : (x{:}\mathsf{Nat}).\mathsf{p} \to \mathsf{r} : (y{:}x < 10).G'$$

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

Certifying Data in MPST

Projection - Data Dependencies

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

$$G = \mathsf{p} \to \mathsf{q} : (x{:}\mathsf{Nat}).\mathsf{p} \to \mathsf{r} : (y{:}x < 10).G'$$

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

$$G \upharpoonright \mathsf{p} \stackrel{?}{=} \mathsf{q}!(x{:}\mathsf{Nat}); \mathsf{r}!(y{:}\Sigma x{:}\mathsf{Nat}.x < 10); (G' \upharpoonright \mathsf{p})$$

Certifying Data in MPST

Projection - Data Dependencies

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

$$G = \mathsf{p} \to \mathsf{q} : (x{:}\mathsf{Nat}).\mathsf{p} \to \mathsf{r} : (y{:}x < 10).G'$$

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

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

Certifying Data in MPST

Projection - Data Dependencies

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

$$G = \mathsf{p} \to \mathsf{q} : (x{:}\mathsf{Nat}).\mathsf{p} \to \mathsf{r} : (y{:}x < 10).G'$$

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

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

Proposed solution:

- Use Σ-types to existentially quantify on the receiver side.
- Use singletons to preserve dependencies on the sender side.

Certifying Data in MPST

Projection - Data Dependencies

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

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

Certifying Data in MPST

Projection - Data Dependencies

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

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

How do we match the input and output types?

Certifying Data in MPST

Projection - Data Dependencies

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

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

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

Certifying Data in MPST

Projection - Data Dependencies

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

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

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

$$\frac{\Psi \vdash M : b}{\Psi \vdash \mathcal{S}(M) \leq b}$$

Certifying Data in MPST

Projection - Data Dependencies

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

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

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

$$\frac{\Psi \vdash M : b}{\Psi \vdash \mathcal{S}(M) \leq b} \qquad \frac{\Psi \vdash \tau \leq \tau' \qquad \Psi, x{:}\tau' \vdash T \leq T'}{\Psi \vdash \mathsf{p}?(x{:}\tau).T \leq \mathsf{p}?(x{:}\tau').T'}$$

$$\frac{\Psi \vdash \tau' \leq \tau \qquad \Psi, x{:}\tau \vdash T \leq T'}{\Psi \vdash \mathsf{p}!(x{:}\tau).T \leq \mathsf{p}!(x{:}\tau').T'}$$

Certifying Data in MPST

Projection

- 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 $\dagger$ 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.

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

$$\mathsf{s} \to \mathsf{r} : (x : \tau).G' \upharpoonright \mathsf{p} \quad = \quad \begin{cases} \end{cases}$$

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

$$\mathsf{s} \to \mathsf{r} : (x : \tau).G' \upharpoonright \mathsf{p} \quad = \quad \begin{cases} \mathsf{r}!(x{:}(CTB(x{:}\tau)); (G' \upharpoonright \mathsf{p}) & \text{if } \mathsf{p} = \mathsf{s} \\ \end{cases}$$

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

$$\mathsf{s} \to \mathsf{r} : (x : \tau).G' \upharpoonright \mathsf{p} \quad = \quad \begin{cases} \mathsf{r}!(x{:}(CTB(x{:}\tau)); (G' \upharpoonright \mathsf{p}) & \text{if } \mathsf{p} = \mathsf{s} \\ \mathsf{s}?(x{:}(CTB(x{:}\tau))^{\dagger}); (G' \upharpoonright \mathsf{p}) & \text{if } \mathsf{p} = \mathsf{r} \\ G' \upharpoonright \mathsf{p} & \text{otherwise} \end{cases}$$

Certifying Data in MPST

Projection

Recall our initial example:

$$
\begin{aligned}
G_{BM} \quad = \quad & \mathsf{B} \to \mathsf{AirMalta} : (d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}). \\
& \mathsf{AirMalta} \to \mathsf{B} : (f{:}\mathsf{String}). \\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}(d_1), \mathsf{Int}(d_2)). \\
& \mathsf{Booking} \to \mathsf{B} : (opt{:}\mathsf{List\ String}). \\
& \mathsf{B} \to \mathsf{Booking} : (\Sigma s{:}\mathsf{String}.s \in opt). \\
& \mathsf{Booking} \to \mathsf{B} : (r{:}\mathsf{Int}). \\
& \mathsf{B} \to \mathsf{COST} : (\mathsf{String}(f), \mathsf{Int}(r)). \\
& \mathsf{COST} \to \mathsf{B} : (\mathsf{ok} : \mathbf{end}, \mathsf{nok} : \mathbf{end})
\end{aligned}
$$

Certifying Data in MPST

Projection

Recall our initial example:

$$
\begin{aligned}
G_{BM} \quad = \quad & \mathsf{B} \to \mathsf{AirMalta} : (d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}). \\
& \mathsf{AirMalta} \to \mathsf{B} : (f{:}\mathsf{String}). \\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}(d_1), \mathsf{Int}(d_2)). \\
& \mathsf{Booking} \to \mathsf{B} : (opt{:}\mathsf{List\ String}). \\
& \mathsf{B} \to \mathsf{Booking} : (\Sigma s{:}\mathsf{String}.s \in opt). \\
& \mathsf{Booking} \to \mathsf{B} : (r{:}\mathsf{Int}). \\
& \mathsf{B} \to \mathsf{COST} : (\mathsf{String}(f), \mathsf{Int}(r)). \\
& \mathsf{COST} \to \mathsf{B} : (\mathsf{ok} : \mathbf{end}, \mathsf{nok} : \mathbf{end})
\end{aligned}
$$

$$
G_{BM} \upharpoonright \mathsf{B} = \quad \mathsf{AirMalta}!(d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}).\mathsf{AirMalta}?(f{:}\mathsf{String}).
$$

Certifying Data in MPST

Projection

Recall our initial example:

$$
\begin{aligned}
G_{BM} \quad = \quad & \mathsf{B} \to \mathsf{AirMalta} : (d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}). \\
& \mathsf{AirMalta} \to \mathsf{B} : (f{:}\mathsf{String}). \\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}(d_1), \mathsf{Int}(d_2)). \\
& \mathsf{Booking} \to \mathsf{B} : (opt{:}\mathsf{List\ String}). \\
& \mathsf{B} \to \mathsf{Booking} : (\Sigma s{:}\mathsf{String}.s \in opt). \\
& \mathsf{Booking} \to \mathsf{B} : (r{:}\mathsf{Int}). \\
& \mathsf{B} \to \mathsf{COST} : (\mathsf{String}(f), \mathsf{Int}(r)). \\
& \mathsf{COST} \to \mathsf{B} : (\mathsf{ok} : \mathbf{end}, \mathsf{nok} : \mathbf{end})
\end{aligned}
$$

$$
\begin{aligned}
G_{BM} \upharpoonright \mathsf{B} = \quad & \mathsf{AirMalta}!(d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}).\mathsf{AirMalta}?(f{:}\mathsf{String}). \\
& \mathsf{Book}!(\Sigma d_1'{:}\mathit{String}(d_1), d_2'{:}\mathsf{Int}(d_2).\mathsf{String}(d_1'), \mathsf{Int}(d_2')).
\end{aligned}
$$

Certifying Data in MPST

Projection

Recall our initial example:

$$
\begin{aligned}
G_{BM} \quad = \quad & \mathsf{B} \to \mathsf{AirMalta} : (d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}).\\
& \mathsf{AirMalta} \to \mathsf{B} : (f{:}\mathsf{String}).\\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}(d_1), \mathsf{Int}(d_2)).\\
& \mathsf{Booking} \to \mathsf{B} : (\mathit{opt}{:}\mathsf{List\ String}).\\
& \mathsf{B} \to \mathsf{Booking} : (\Sigma s{:}\mathsf{String}.s \in \mathit{opt}).\\
& \mathsf{Booking} \to \mathsf{B} : (r{:}\mathsf{Int}).\\
& \mathsf{B} \to \mathsf{COST} : (\mathsf{String}(f), \mathsf{Int}(r)).\\
& \mathsf{COST} \to \mathsf{B} : (\mathsf{ok} : \mathbf{end}, \mathsf{nok} : \mathbf{end})
\end{aligned}
$$

$$
\begin{aligned}
G_{BM} \upharpoonright \mathsf{B} = \quad & \mathsf{AirMalta}!(d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}).\mathsf{AirMalta}?(f{:}\mathsf{String}).\\
& \mathsf{Book}!(\Sigma d_1'{:}\mathit{String}(d_1), d_2'{:}\mathsf{Int}(d_2).\mathsf{String}(d_1'), \mathsf{Int}(d_2')).\\
& \mathsf{Book}?(\mathit{opt}{:}\mathsf{List\ String}).\mathsf{Booking}!(\Sigma s{:}\mathsf{String}.s \in \mathit{opt}).
\end{aligned}
$$

Certifying Data in MPST

Projection

Recall our initial example:

$$
\begin{aligned}
G_{BM} \quad = \quad & \mathsf{B} \to \mathsf{AirMalta} : (d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}). \\
& \mathsf{AirMalta} \to \mathsf{B} : (f{:}\mathsf{String}). \\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}(d_1), \mathsf{Int}(d_2)). \\
& \mathsf{Booking} \to \mathsf{B} : (opt{:}\mathsf{List\ String}). \\
& \mathsf{B} \to \mathsf{Booking} : (\Sigma s{:}\mathsf{String}.s \in opt). \\
& \mathsf{Booking} \to \mathsf{B} : (r{:}\mathsf{Int}). \\
& \mathsf{B} \to \mathsf{COST} : (\mathsf{String}(f), \mathsf{Int}(r)). \\
& \mathsf{COST} \to \mathsf{B} : (\mathsf{ok} : \mathbf{end}, \mathsf{nok} : \mathbf{end})
\end{aligned}
$$

$$
\begin{aligned}
G_{BM} \upharpoonright \mathsf{B} = \quad & \mathsf{AirMalta}!(d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}).\mathsf{AirMalta}?(f{:}\mathsf{String}). \\
& \mathsf{Book}!(\Sigma d_1'{:}\mathit{String}(d_1), d_2'{:}\mathsf{Int}(d_2).\mathsf{String}(d_1'), \mathsf{Int}(d_2')). \\
& \mathsf{Book}?(opt{:}\mathsf{List\ String}).\mathsf{Booking}!(\Sigma s{:}\mathsf{String}.s \in opt). \\
& \mathsf{Book}?(r{:}\mathsf{Int}).\mathsf{COST}!(\Sigma f'{:}\mathsf{String}(f), r'{:}\mathsf{Int}(r).\mathsf{String}(f'), \mathsf{Int}(r')). \\
& \ldots
\end{aligned}
$$

Certifying Data in MPST

Projection – Methodology

To summarize:

- Given a well-formed global type G
- Project well-formed local types $\{G \upharpoonright \mathsf{p}_i\}_{i \in \mathsf{pid}(G)}$, preserving dependencies.
- Find endpoint processes that satisfy each of $G \upharpoonright \mathsf{p}_i$.

Results

Type Safety

Typing Judgment

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

Process P uses its session channels as specified in Δ; shared channels as specified in Γ and message variables as in Ψ.

Results

Type Safety

Typing Judgment

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

Process P uses its session channels as specified in Δ; shared channels as specified in Γ and message variables as in Ψ.

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

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

Process P uses its session channels as specified in Δ; shared channels as specified in Γ and message variables as in Ψ.

Subject Transition

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

Results

Type Safety

Typing Judgment

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

Process P uses its session channels as specified in Δ; shared channels as specified in Γ and message variables as in Ψ.

Subject Transition

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

Results

Type Safety

Typing Judgment

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

Process P uses its session channels as specified in Δ; shared channels as specified in Γ and message variables as in Ψ.

Subject Transition

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

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

Results

Type Safety

Typing Judgment

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

Process P uses its session channels as specified in Δ; shared channels as specified in Γ and message variables as in Ψ.

Subject Transition

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

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

(b) If α is an input or a branching then for all Δ', 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 $\Diamond_{\mathsf{p}}\tau$ 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 $\Diamond_{\mathsf{p}}\tau$ and erasable $[\tau]$.
- A message $M : \Diamond_{\mathsf{p}}[\tau]$ denotes:
  - A term M signed by participant p (public key infrastructure).
  - Denotes the existence of a proof of τ.
  - May not itself contain such a proof (erasure of []'d terms).

Applications

Signed Messages

- $\Diamond_{\mathsf{p}}\tau$ is a strong (role-indexed) monad.

Applications

Signed Messages

- $\Diamond_{\mathsf{p}}\tau$ is a strong (role-indexed) monad.
- "Diamonds are forever" – From $M : \Diamond_{\mathsf{p}}\tau$ cannot extract $N : \tau$.

Applications

Signed Messages

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

Applications

Signed Messages

- $\Diamond_{\mathsf{p}}\tau$ is a strong (role-indexed) monad.
- "Diamonds are forever" – From $M : \Diamond_{\mathsf{p}}\tau$ 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 $\mathsf{p} \to \mathsf{q} : (\Diamond_{\mathsf{p}'}\tau).G' \sqsubseteq G$ then $\mathsf{p}' = \mathsf{p}$ or $\mathsf{r} \to \mathsf{p} : (\Diamond_{\mathsf{p}'}\tau).G'' \sqsubseteq G$ "earlier".

Applications

Signed Messages

- $\Diamond_{\mathsf{p}}\tau$ is a strong (role-indexed) monad.
- "Diamonds are forever" – From $M : \Diamond_{\mathsf{p}}\tau$ 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

$$
\begin{array}{lll}
G_{BM} = & \mathsf{B} \to \mathsf{AirMalta} : (d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}). & \text{Dest. + date} \\
& \mathsf{AirMalta} \to \mathsf{B} : (f{:}\mathsf{String}). & \text{Flight Info.} \\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}(d_1), \mathsf{Int}(d_2)). & \text{Dest. + date} \\
& \mathsf{Booking} \to \mathsf{B} : (opt{:}\mathsf{List\ String}). & \text{Options} \\
& \mathsf{B} \to \mathsf{Booking} : (\Sigma s{:}\mathsf{String}.s \in opt). & \text{Choose} \\
& \mathsf{Booking} \to \mathsf{B} : (r{:}\mathsf{Int}). & \text{Receipt} \\
& \mathsf{B} \to \mathsf{COST} : (\mathsf{String}, \mathsf{Int}). & \text{Expense claims} \\
& \mathsf{COST} \to \mathsf{B} : (\mathsf{ok} : \mathbf{end}; & \text{ok or not} \\
& \qquad\qquad\qquad \mathsf{nok} : \mathbf{end}) &
\end{array}
$$

Applications

Digital Certificates

$$
\begin{array}{lll}
G_{BM} = & \mathsf{B} \to \mathsf{AirMalta} : (d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}). & \text{Dest. + date} \\
& \mathsf{AirMalta} \to \mathsf{B} : (f{:}\Diamond_{AM}\mathsf{String}). & \text{Flight Info.} \\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}(d_1), \mathsf{Int}(d_2)). & \text{Dest. + date} \\
& \mathsf{Booking} \to \mathsf{B} : (opt{:}\mathsf{List\ String}). & \text{Options} \\
& \mathsf{B} \to \mathsf{Booking} : (\Sigma s{:}\mathsf{String}.s \in opt). & \text{Choose} \\
& \mathsf{Booking} \to \mathsf{B} : (r{:}\Diamond_{BK}\mathsf{Int}). & \text{Receipt} \\
& \mathsf{B} \to \mathsf{COST} : (\Diamond_{AM}\mathsf{String}, \Diamond_{BK}\mathsf{Int}). & \text{Expense claims} \\
& \mathsf{COST} \to \mathsf{B} : (\mathsf{ok} : \mathbf{end}; & \text{ok or not} \\
& \qquad\qquad\qquad \mathsf{nok} : \mathbf{end}) &
\end{array}
$$

Trust no one, sign “everything”.

Applications

Digital Certificates

$$
\begin{array}{lll}
G_{BM} = & \mathsf{B} \to \mathsf{AirMalta} : (d_1{:}\mathsf{String}, d_2{:}\mathsf{Int}). & \text{Dest. + date} \\
& \mathsf{AirMalta} \to \mathsf{B} : (f{:}\mathsf{String}, \Diamond_{AM}[\mathsf{String}]). & \text{Flight Info.} \\
& \mathsf{B} \to \mathsf{Booking} : (\mathsf{String}(d_1), \mathsf{Int}(d_2)). & \text{Dest. + date} \\
& \mathsf{Booking} \to \mathsf{B} : (opt{:}\mathsf{List\ String}). & \text{Options} \\
& \mathsf{B} \to \mathsf{Booking} : (\Sigma s{:}\mathsf{String}.s \in opt). & \text{Choose} \\
& \mathsf{Booking} \to \mathsf{B} : (r{:}\mathsf{Int}, \Diamond_{BK}[\mathsf{Int}]). & \text{Receipt} \\
& \mathsf{B} \to \mathsf{COST} : (\Diamond_{AM}[\mathsf{String}], \Diamond_{BK}[\mathsf{Int}]). & \text{Expense claims} \\
& \mathsf{COST} \to \mathsf{B} : (\mathsf{ok} : \mathbf{end}; & \text{ok or not} \\
& \qquad\qquad\qquad \mathsf{nok} : \mathbf{end}) &
\end{array}
$$

Trust a bit more, allow for erasure at runtime.

Applications

Digital Certificates

A slight variation:

$$
\begin{array}{lcll}
G_{BM} & = & \text{Booking} \to \text{B} : (opt{:}\text{List String}). & \text{Options} \\
 & & \text{B} \to \text{Booking} : (c{:}\Sigma s{:}\text{String}.s \in opt). & \text{Choose} \\
 & & \text{Booking} \to \text{B} : (r{:}\Sigma i{:}\text{Int}.\text{receipt}(\pi_1(c))). & \text{Better Receipt} \\
 & & \text{B} \to \text{COST} : (\text{Int}(\pi_1(r))). & \text{Claim} \\
 & & \ldots &
\end{array}
$$

Applications

Digital Certificates

A slight variation:

$$
\begin{array}{lcll}
G_{BM} & = & \mathsf{Booking} \to \mathsf{B} : (opt{:}\mathsf{List\ String}). & \text{Options} \\
 & & \mathsf{B} \to \mathsf{Booking} : (c{:}\Sigma s{:}\mathsf{String}.s \in opt). & \text{Choose} \\
 & & \mathsf{Booking} \to \mathsf{B} : (r{:}\Sigma i{:}\mathsf{Int}.\mathsf{receipt}(\pi_1(c))). & \text{Better Receipt} \\
 & & \mathsf{B} \to \mathsf{COST} : (\mathsf{Int}(\pi_1(r))). & \text{Claim} \\
 & & \ldots &
\end{array}
$$

$$
G_{BM} \upharpoonright \mathsf{COST} = \mathsf{B}?(\Sigma opt{:}\ldots, c{:}\ldots, r{:}(\Sigma i{:}\mathsf{Int}.\mathsf{receipt}(\pi_1(c))).\mathsf{Int}(\pi_1(r)))
$$

Applications

Digital Certificates

A slight variation:

$$
\begin{array}{lcll}
G_{BM} & = & \mathsf{Booking} \to \mathsf{B} : (opt{:}\mathsf{List\ String}). & \text{Options} \\
 & & \mathsf{B} \to \mathsf{Booking} : (c{:}\Sigma s{:}\mathsf{String}.s \in opt). & \text{Choose} \\
 & & \mathsf{Booking} \to \mathsf{B} : (r{:}\Sigma i{:}\mathsf{Int}.[\mathsf{receipt}(\pi_1(c))]). & \text{Better Receipt} \\
 & & \mathsf{B} \to \mathsf{COST} : (\mathsf{Int}(\pi_1(r))). & \text{Claim} \\
 & & \ldots &
\end{array}
$$

Applications

Digital Certificates

A slight variation:

$$
\begin{array}{lll}
G_{BM} = & \text{Booking} \to \text{B} : (opt{:}\text{List String}). & \text{Options} \\
& \text{B} \to \text{Booking} : (c{:}\Sigma s{:}\text{String}.s \in opt). & \text{Choose} \\
& \text{Booking} \to \text{B} : (r{:}\Sigma i{:}\text{Int}.[\text{receipt}(\pi_1(c))]). & \text{Better Receipt} \\
& \text{B} \to \text{COST} : (\text{Int}(\pi_1(r))). & \text{Claim} \\
& \ldots &
\end{array}
$$

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

$$G_{BM}^{\downarrow} \upharpoonright \text{COST} = \text{B?}(\text{Int})$$

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