# Proof-Carrying Code with Dependent Session Types

Bernardo Toninho
[with Luís Caires and Frank Pfenning]

Center for Informatics and Information Technology (CITI)
Computer Science Department
FCT-UNL & Carnegie Mellon University

Betty Meeting 2013

# Background

## Challenges

Reasoning about distributed software systems in the presence of complex requirements is necessary but hard:

- Functionality
- Integrity
- Deadlock/Livelock Freedom

# Background

## Challenges

Reasoning about distributed software systems in the presence of complex requirements is necessary but hard:

- Functionality
- Integrity
- Deadlock/Livelock Freedom

## Key Issues

- How can we express the properties of interest?
- How can we enforce them using static (compile-time) and dynamic (run-time) methods?

# Approach

## Session Types

- Types are behavioral specifications of communication protocols.
- Statically checkable (simple) protocols.
- Safety “for free”.

# Approach

## Session Types

- Types are behavioral specifications of communication protocols.
- Statically checkable (simple) protocols.
- Safety "for free".

## Logic

- Propositions (types) talk about (complex) properties.
- In intuitionistic logic, propositions (types) talk about proofs.

# Our Work

- A logical interpretation of dependent session types:
  - Types can talk about communicated data.
  - Types can talk about rich properties and proofs.
  - Programs communicate data and proofs about the data!

# Our Work

- A logical interpretation of dependent session types:
  - Types can talk about communicated data.
  - Types can talk about rich properties and proofs.
  - Programs communicate data and proofs about the data!
- Two extensions, based on logic:
  - Omit proofs at runtime – Proof Irrelevance
  - Digital certificates – Affirmation

# Our Work

- A logical interpretation of dependent session types:
  - Types can talk about communicated data.
  - Types can talk about rich properties and proofs.
  - Programs communicate data and proofs about the data!
- Two extensions, based on logic:
  - Omit proofs at runtime – Proof Irrelevance
  - Digital certificates – Affirmation
- Applications:
  - Statically certified distributed computing
  - Runtime system for proof-carrying/certified distributed software.

# Types

## Simple Session Types

$$S, T \quad ::= \quad \mathbf{1} \qquad \text{Stop}$$

# Types

## Simple Session Types

$$
\begin{array}{rcll}
S, T & ::= & \mathbf{1} & \text{Stop} \\
 & | & T \otimes S & \text{Output}
\end{array}
$$

# Types

## Simple Session Types

$$
\begin{array}{rcll}
S, T & ::= & \mathbf{1} & \text{Stop} \\
 & \mid & T \otimes S & \text{Output} \\
 & \mid & T \multimap S & \text{Input}
\end{array}
$$

# Types

## Simple Session Types

$$
\begin{array}{lcll}
S, T & ::= & \mathbf{1} & \text{Stop} \\
 & \mid & T \otimes S & \text{Output} \\
 & \mid & T \multimap S & \text{Input} \\
 & \mid & !S & \text{Replication}
\end{array}
$$

# Types

## Simple Session Types

| | | | |
|---|---|---|---|
| $S, T$ | $::=$ | $\mathbf{1}$ | Stop |
| | $\mid$ | $T \otimes S$ | Output |
| | $\mid$ | $T \multimap S$ | Input |
| | $\mid$ | $!S$ | Replication |
| | $\mid$ | $\tau$ | Base Types |
| | $\mid$ | $\ldots$ | |

# Types

## Simple Session Types

$$
\begin{array}{rcll}
S, T & ::= & \mathbf{1} & \text{Stop} \\
 & \mid & T \otimes S & \text{Output} \\
 & \mid & T \multimap S & \text{Input} \\
 & \mid & !S & \text{Replication} \\
 & \mid & \tau & \text{Base Types} \\
 & \mid & \ldots &
\end{array}
$$

## Typing Judgment

$$x_1 : S_1, \ldots, x_k : S_k \vdash P :: m{:}S$$

Process $P$, when composed with systems providing behavior $S_j$ at $x_j$, yields a deadlock-free system providing behavior $S$ at $m$.

# Example - PDF Indexer

A persistent PDF indexing service:

$$\text{index} \triangleq\ !(\text{file} \multimap (\text{file} \otimes \mathbf{1}))$$

A persistent service that receives a file (a pdf) and outputs a file (an indexed version of the pdf).

# Example - PDF Indexer

A persistent PDF indexing service:

$$\text{index} \triangleq\ !(\text{file} \multimap (\text{file} \otimes \mathbf{1}))$$

A persistent service that receives a file (a pdf) and outputs a file (an indexed version of the pdf).

## Remark

- Type doesn't specify the functional behavior, just communication!
  - "Persistent service that inputs a file and outputs a file"
- "Just trust me on it" – Not reasonable in a distributed setting.

# Types Revisited

Basic types $\tau$ are now a dependent type theory:

- Types as arbitrary properties (predicates, relations)
- Terms as proofs of the properties

# Types Revisited

Basic types $\tau$ are now a dependent type theory:

- Types as arbitrary properties (predicates, relations)
- Terms as proofs of the properties

## Dependent Session Types

$$
\begin{array}{rcll}
T, S & ::= & \forall x{:}\tau.A & \text{Input } M{:}\tau \text{ continue as } A\{M/x\} \\
 & \mid & \exists x{:}\tau.A & \text{Output } M{:}\tau \text{ continue as } A\{M/x\}
\end{array}
$$

# Types Revisited

Basic types $\tau$ are now a dependent type theory:

- Types as arbitrary properties (predicates, relations)
- Terms as proofs of the properties

## Dependent Session Types

| | | | |
|---|---|---|---|
| $T, S$ | $::=$ | $\forall x{:}\tau.A$ | Input $M{:}\tau$ continue as $A\{M/x\}$ |
| | $\mid$ | $\exists x{:}\tau.A$ | Output $M{:}\tau$ continue as $A\{M/x\}$ |
| $\tau$ | $::=$ | $[\tau]$ | Erasable proof of $\tau$ |
| | $\mid$ | $\Diamond_K \tau$ | Principal $K$ produces a certificate for a proof of $\tau$ |

Certificates assume a public key infrastructure.

# Example - PDF Indexer Revisited

- A persistent PDF indexer:

$$\text{index} \triangleq\ !(\text{file} \multimap (\text{file} \otimes \mathbf{1}))$$

# Example - PDF Indexer Revisited

- A persistent PDF indexer:

$$\text{index} \triangleq !(\text{file} \multimap (\text{file} \otimes \mathbf{1}))$$

- A certifying indexer:

$$\text{index}_1 \triangleq !(\forall f : \text{file.pdf}(f) \multimap \exists g : \text{file.pdf}(g) \otimes \text{agree}(f, g) \otimes \mathbf{1})$$

# Example - PDF Indexer Revisited

- A persistent PDF indexer:

$$\text{index} \triangleq !(\text{file} \multimap (\text{file} \otimes \mathbf{1}))$$

- A certifying indexer:

$$\text{index}_1 \triangleq !(\forall f : \text{file}.\text{pdf}(f) \multimap \exists g : \text{file}.\text{pdf}(g) \otimes \text{agree}(f, g) \otimes \mathbf{1})$$

- A trusted indexer – No proofs at runtime:

$$\text{index}_2 \triangleq !(\forall f : \text{file}.[\text{pdf}(f)] \multimap \exists g : \text{file}.[\text{pdf}(g)] \otimes [\text{agree}(f, g)] \otimes \mathbf{1})$$

# Example - PDF Indexer Revisited

- A persistent PDF indexer:

$$\text{index} \triangleq !(\text{file} \multimap (\text{file} \otimes \mathbf{1}))$$

- A certifying indexer:

$$\text{index}_1 \triangleq !(\forall f : \text{file.pdf}(f) \multimap \exists g : \text{file.pdf}(g) \otimes \text{agree}(f, g) \otimes \mathbf{1})$$

- A trusted indexer – No proofs at runtime:

$$\text{index}_2 \triangleq !(\forall f : \text{file.}[\text{pdf}(f)] \multimap \exists g : \text{file.}[\text{pdf}(g)] \otimes [\text{agree}(f, g)] \otimes \mathbf{1})$$

- A liable indexer – Signed certificate transmitted:

$$\text{index}_2 \triangleq !(\forall f : \text{file.}[\text{pdf}(f)] \multimap \exists g : \text{file.}[\text{pdf}(g)] \otimes \Diamond_I[\text{agree}(f, g)] \otimes \mathbf{1})$$

# Technical Results

We obtain the following soundness results for our system:

- Type Preservation - “Internal actions don’t alter types”.
- Progress - “Systems can always take actions”.

# Technical Results

We obtain the following soundness results for our system:

- Type Preservation - “Internal actions don’t alter types”.
- Progress - “Systems can always take actions”.

In *practice*, this means:

- **Protocol Fidelity** - “Protocols are guaranteed to be played out”.
- **Deadlock and Livelock Absence** - “Systems don’t get stuck”.
- **Logical assertions in protocols always hold**.

# Concluding Remarks

## Contributions

- A logically motivated system of proof-carrying communication.
- Exploit the logical foundation:
  - Communicating proofs (explicit or implicit through proof irrelevance)
  - Digital signatures (implicit or explicit via affirmation)
- Clean integration of reasoning and computation.

# Concluding Remarks

## Contributions

- A logically motivated system of proof-carrying communication.
- Exploit the logical foundation:
  - Communicating proofs (explicit or implicit through proof irrelevance)
  - Digital signatures (implicit or explicit via affirmation)
- Clean integration of reasoning and computation.

## Future Work

- Practical language design considerations.
- Reasoning about processes, not just communicated data.