A Logical Foundation for Proof-Carrying Communicating Processes

Bernardo Toninho
[with Luís Caires and Frank Pfenning]

Computer Science Department
Carnegie Mellon University
Universidade Nova de Lisboa

Speaking Skills Talk
Concurrent and Distributed Systems

- *Everything* is becoming more and more concurrent!
- Building these systems is hard.
- Building these systems *correctly* is harder.
- *Reasoning* about these systems is even harder.

Why is it hard?

- New problems arise: deadlocks, livelocks, security, etc.
- Compositional reasoning is generally hard to do in this setting (if possible).
- Testing isn’t a reasonable tool (conditions hard to replicate).
Correctness - What does it mean?
- System doesn’t deadlock.
- System eventually produces some result.
- The result is correct.

What can be ensured *statically* today?
- Absence of deadlocks (mostly in very high-level calculi)
- Progress (same as above, typically hard to guarantee).
- Simple properties of communicated data (hard to combine with points above).
How? Sessions

Structuring Communication

- Communication without structure is hard to reason about.
- Structure communication around the concept of sessions.
- Predetermined sequences of interactions along a (session) channel:
  - “Input a number, output a string and terminate”.
  - “Either output or input a number”.
  - “Offer a choice between outputting or inputting a number”.

Sessions

- Specify communication behavior as sessions.
- Check that programs obey specification (session fidelity).

Sounds a lot like type-checking...
Session Types

- Types are descriptions of communication behavior.
- A way of guaranteeing communication discipline, statically.
- What behavioral specifications do we typically need?

Base Types

\[
\tau\quad \text{A basic data value of type } \tau
\]

Any basic type we need (integers, strings, etc.) - omit $ for readability.

Input

\[
A \rightarrow B \quad \text{Input a session of type } A \text{ and continue as } B
\]

A (too) simple bank service: \( \text{bank}_1 \triangleq \text{string} \rightarrow (\text{nat} \rightarrow 1) \)
A ⊗ B  
Output a session of type $A$ and continue as $B$

A simple bank service: $\text{bank}_2 \triangleq \text{string} \rightarrow (\text{nat} \rightarrow (\text{nat} \otimes 1))$

A PDF indexing service: $\text{index}_1 \triangleq \text{file} \rightarrow (\text{file} \otimes 1)$

Persistence or Replication

A session behavior that can be accessed multiple times:

!$A$  
Replicated session of type $A$

A reusable bank: $\text{bank}_3 \triangleq !(\text{string} \rightarrow (\text{nat} \rightarrow (\text{nat} \otimes 1)))$

A reusable indexer: $\text{index}_2 \triangleq !(\text{file} \rightarrow (\text{file} \otimes 1))$

Persistent storage: $\text{store}_1 \triangleq !(\text{file} \rightarrow !(\text{file} \otimes 1))$
Branch and Choice

\[ A \& B \quad \text{Offer choice between a session of type } A \text{ and } B \]

\[ A \oplus B \quad \text{Either a session of type } A \text{ or } B \text{ (no choice for client)} \]

Money withdrawal: \[ wd \triangleq \text{nat} \rightarrow ((\text{money} \otimes 1) \oplus (\text{string} \otimes 1)) \]

A more useful bank: \[ \text{bank}_4 \triangleq !(\text{string} \rightarrow (\text{dep} \& \text{wd})) \]
### Session Types

- **A → B**: Input a session of type \( A \) and continue as \( B \)
- **A ⊗ B**: Output a session of type \( A \) and continue as \( B \)
- **A & B**: Choice between a session of type \( A \) or \( B \)
- **A ⊕ B**: Either a session of type \( A \) or \( B \)
- **!A**: A persistent session of type \( A \)
- **1**: Terminated session
- **$\tau$**: Base types

### Typing Judgment

\[
\Gamma, \underbrace{u_1:A_1, \ldots, u_m:A_m; x_1:A_1, \ldots, x_n:A_n}_\Delta \vdash P :: x:A
\]
Session Types and Intuitionistic Linear Logic

- It's possible to interpret session types as linear logic propositions.
- Linear logic proofs as process typing derivations.
- Rules of logic as process typing rules.

Why should we care?

- Results from logic carry over to session typings:
  - Progress, session fidelity, type preservation “for free”.
- Extending the system with new ideas from logic becomes not only possible, but fairly simple.
Limitations

- No real way of talking about the outcome of a session:

\[ \text{index}_2 \triangleq !(\text{file} \rightarrow (\text{file} \otimes 1)) \]

Typing doesn’t ensure anything about what the output file is.

- No way of specifying or verifying (functional) correctness statically.

- “Just trust me on it” – Unreasonable in a distributed setting.

Can we do something about this? **Yes**, by using logic.
The language of basic terms of type $\tau$ is not specified.

We will consider types $\tau$ from a (dependent) type theory.

- Types can depend on terms
- In practice, types denote *properties*, terms denote *proofs*.

Generalize idioms $\tau \to A$ and $\tau \otimes A$.

**Generalizing the Interpretation - Term Input**

$$ \forall x:\tau. A \quad \text{Input a term } M:\tau \text{ and proceed as } A\{M/x\} $$

A reusable indexer: $\text{index}_2 \triangleq !((\text{file} \to (\text{file} \otimes 1)))$

A better indexer: $\text{index}_3 \triangleq !((\forall f:\text{file.pdf}(f) \to \text{file} \otimes 1))$

Indexer now requires a *proof* that the file is a pdf!
Generalizing the Interpretation - Term Output

\[ \exists x : \tau . A \] Output a term \( M : \tau \) and proceed as \( A\{M/x\} \)

A better indexer:
\[ \text{index}_3 \triangleq ! (\forall f: \text{file.pdf}(f) \rightarrow \text{file} \otimes 1) \]

Even better:
\[ \text{index}_4 \triangleq ! (\forall f: \text{file.pdf}(f) \rightarrow \exists g: \text{file.pdf}(g) \otimes 1) \]

Require proof that file is a pdf, supply a proof that output is also a pdf.
“Glossary” of types:

- \( \text{uid}(s) \): \( s \) represents a user id
- \( \text{deposit}(s, n) \): Deposit for user \( s \) of \( n \) dollars
- \( \text{receipt}(s, n) \): Deposit receipt for user \( s \) of \( n \) dollars
- \( \text{balance}(s, n) \): User \( s \) has \( n \) dollars

A bank service (deposits and balance inquiries):

\[
\text{bank}_5 \triangleq !(\forall s:\text{string}.\text{uid}(s) \rightarrow
(\forall n:\text{nat}.\text{deposit}(s, n) \rightarrow (\text{receipt}(s, n) \otimes 1)) \ & \ (\exists m:\text{nat}.\text{balance}(s, m) \otimes 1)))
\]

After the login, bank offers deposit and balance inquiry services.
“Glossary” of types:

- `uid(s)` represents a user id
- `deposit(s, n)` Deposit for user `s` of `n` dollars
- `receipt(s, n)` Deposit receipt for user `s` of `n` dollars
- `balance(s, n)` User `s` has `n` dollars

ATM service (only deposit portion):

\[
\text{atm} \triangleq \forall s:\text{string}. \text{uid}(s) \rightsquigarrow \\
(\forall n:\text{nat}. \text{deposit}(s, n) \rightsquigarrow \exists m:\text{nat}. \\
\exists p: (n - 2 \leq m \leq n). \text{receipt}(s, m) \otimes 1)
\]

`atm` interfaces with bank, charging at most $2$ for a deposit.
Start from a logical interpretation of session types with data.

Extend the interpretation with first-order type constructors:
- \(\forall x: \tau. A\) – Input a term \(M : \tau\), continue as \(A\{M/x\}\).
- \(\exists x: \tau. A\) – Output a term \(M : \tau\), continue as \(A\{M/x\}\).

A dependently-typed term language enables communication of proof objects.

Session types can capture rich properties of communicated data, witnessed by exchange of explicit proof obligations.
Often we care about the existence of proofs, but not about the actual proofs:

- In the indexer example, we can check pdf(g) ourselves.
- Some proofs are not particularly informative (e.g. atm service).

Solution: Employ *Proof Irrelevance* in the term language.

\[ M : [\tau] - M \text{ is a term of type } \tau \text{ that is computationally irrelevant.} \]

Agree that computationally irrelevant terms will be erased before transmission.

Typing ensures this can be done consistently.
Communicating Proofs

Proof Irrelevance

### Rules

\[
\frac{\psi \ni M : \tau}{\psi \ni [M] : [\tau]} \quad [I] \\
\frac{\psi \ni M : [\tau]}{\psi, x \vdash \tau \ni N : \sigma} \quad [E]
\]

(\(\psi \oplus\) promotes hypotheses \(x \vdash \tau\) to \(x : \tau\))

- Typing guarantees that terms \([M]\) can be erased safely.

### Using Proof Irrelevance

We mark proofs as computationally irrelevant:

\[
\text{index}_4 \triangleq !(\forall f:\text{file.pdf}(f) \rightarrow \exists g:\text{file.pdf}(g) \otimes 1) \\
\text{index}_5 \triangleq !(\forall f:\text{file.[pdf}(f)] \rightarrow \exists g:\text{file.[pdf}(g)] \otimes 1)
\]
A revised atm service:

\[
\text{atm}_2 \triangleq \forall s:\text{string}.\text{uid}(s) \rightarrow \\
(\forall n:\text{nat}.\text{deposit}(s, n) \rightarrow \exists m:\text{nat}. \\
\exists p:[n - 2 \leq m \leq n].\text{receipt}(s, m) \otimes 1)
\]

Proof \(p\) must exist for typechecking, but can be erased at runtime.

A verifying indexer:

\[
\text{index}_6 \triangleq !((\forall f:\text{file}.[\text{pdf}(f)] \rightarrow \exists g:\text{file}.[\text{pdf}(g)] \otimes \\
([\text{agree}(f, g)] \otimes 1))
\]

\(\text{agree}(f, g)\) if \(f\) and \(g\) differ only in the index.
Completely erasing proofs can be too extreme (from no trust to complete trust).

Since proofs can be omitted at runtime, clients may require a certificate that affirms the existence of proofs:
- Indexer certifies the received and sent files agree.
- Bank and ATM certify deposit cost at most 2 dollars.

Solution: Use affirmation (from modal logic) in the term language.

\[ M : \mathcal{K} \tau : \text{Principal } K \text{ affirms property } \tau, \text{ with evidence } M. \]
Affirmation

\[
\frac{\Psi \vdash M : \tau}{\Psi \vdash \langle M : \tau \rangle_K : K \tau} \quad \text{(affirms)}
\]

\[
\frac{\Psi \vdash M : K \tau}{\Psi \vdash M : \diamond K \tau}
\]

\[
\frac{\Psi \vdash M : \diamond K \tau, x : \tau \vdash N : K \sigma}{\Psi \vdash \text{let } \langle x : \tau \rangle_K = M \text{ in } N : K \sigma}
\]

- Assume some public key infrastructure to generate these objects.
- \(\langle M : \tau \rangle_K\) can be realized by \(K\)'s signature on \(M : \tau\).
- Internalize judgment as proposition (type) \(\diamond K \tau\).
Communicating Proofs
Digital Signatures and Affirmation

Objects of type $\diamond_K \tau$ denote a witness to property $\tau$, for which $K$ is accountable:

- **PDF indexing service, with indexer X:**
  
  $\text{index}_7 \triangleq \exists (\forall f:\text{file}.[\text{pdf}(f)] \rightarrow \\
  \exists g:\text{file}.[\text{pdf}(g)] \otimes \diamond_X [\text{agree}(g, f)] \otimes 1)$

- **ATM, identified by principal A:**
  
  $\text{atm}_3 \triangleq \forall s:\text{string}.\text{uid}(s) \rightarrow \\
  (\forall n:\text{nat}.\text{deposit}(s, n) \rightarrow \exists m:\text{nat}. \\
  \exists p: \diamond_A [n - 2 \leq m \leq n].\text{receipt}(s, m) \otimes 1)$

- **What can be transmitted when we use the idiom $\diamond_K [\tau]$?**
  
  - $\langle [] : \tau \rangle_K$, a certificate, signed by $K$, affirming $\tau$
  - A proof that $[\tau]$ follows from affirmations by $K$, according to the laws of $\diamond_K$. 

Bernardo Toninho (UNL&CMU)
Speaking Skills Talk
23 / 32
Recall the PDF indexer:

\[ \text{index}_7 \triangleq ! (\forall f : \text{file.}[\text{pdf}(f)] \rightarrow \exists g : \text{file.}[\text{pdf}(g)] \otimes \Diamond_X [\text{agree}(g, f)] \otimes 1) \]

A PDF compression service, with compressor \( C \):

\[ \text{compress} \triangleq ! (\forall f : \text{file.}[\text{pdf}(f)] \rightarrow \exists g : \text{file.}[\text{pdf}(g)] \otimes \Diamond_C [\text{approx}(g, f)] \otimes 1) \]

A composite service, indexing and compression:

\[ \text{ixc} \triangleq ! (\forall f : \text{file.}[\text{pdf}(f)] \rightarrow \exists g : \text{file.}[\text{pdf}(g)] \otimes \Diamond_X \Diamond_C [\text{approx}(g, f)] \otimes 1) \]

Affirmation tracks the principals that need to be trusted!
• Affirmation tracks provenance and info. flow:
  • In general, not the case that $\diamond K\tau \rightarrow \tau$.
  • Needs a form declassification.

• Modelling trust from affirmations:
  • For specific types $\tau$ and principals $K$:

    $$\text{trust}_{K,\tau} : \diamond K\tau \rightarrow \tau$$

• Implementable by stripping signature.
• Erased proofs $[\tau]$ cannot generally be recovered:
  • $\forall [\tau] \rightarrow \tau$
  • $\forall \diamond K[\tau] \rightarrow \tau$
Conclusion and Future Work

Contributions

- A logical foundation for proof-carrying processes:
  - Based on a Curry-Howard interpretation of linear logic
  - First-order connectives specify value and proof communication
  - Proof-passing extension with a type theory

- Additional type theoretic constructs for added flexibility:
  - Proof irrelevance to mark non-communicated proofs.
  - Affirmations for certificates based on digital signatures.

- High-level considerations of trust and liability:
  - Explicit proofs: No trust.
  - Erased proofs: “full” trust.
  - Certificates: Trust with liability for the certifier.
A solid foundation in logic and theory:
- Allows for modular construction and extensibility
- Integrates reasoning (proofs) and computation

Uniform logical integration:
- Proofs (implicit or explicit)
- Affirmations (implicit or explicit signatures)

Logic just makes it all work!
Conclusion and Future Work

Future and Ongoing Work

- Practical language design considerations
- Reasoning further about processes:
  - Observational equivalences
  - Concurrent type theory
- Asynchronous communication
- Polymorphism and parametricity
- ... and more!
Related Work

“Types for Dyadic Interaction”. K. Honda, CONCUR’93.
- First proposal of session-types;
- No “first-order” types;
- Only able to describe basic communication behavior.

“Session Types as Intuitionistic Linear Propositions”. L. Caires, F. Pfenning, CONCUR’11
- First logical interpretation of sessions;
- Basis of our work;
- Only “simple” sessions, no value-passing.
“Correspondence Assertions for Process Synchronization in Concurrent Computation”. E. Bonelli, A. Compagnoni and E. L. Gunther, JFP’05

- Combines “simple” sessions with correspondence assertions;
- No proof passing – Assertions need to be decidable in practice;
- A less flexible approach in general.
A Logical Foundation for Proof-Carrying Communicating Processes

Bernardo Toninho
[with Luís Caires and Frank Pfenning]

Computer Science Department
Carnegie Mellon University
Universidade Nova de Lisboa

Speaking Skills Talk
Example: consolidator implementation

- **Specification**

  \[ \text{ixc} \triangleq ! (\forall f: \text{file}. \ [\text{pdf}(f)]) \]

  \[ \rightarrow \exists g: \text{file}. \ [\text{pdf}(g)] \otimes \Diamond x \Diamond c [\text{approx}(g, f)] \otimes 1 \]

- **Implementation**

  \[ \text{consolidator} = !\text{ixc}(a).a(f_1).a([p_1]). \]

  \[ (\nu b) \text{index}(b).b<f_1>.b([p_1]).b(f_2).b([p_2]).b(q_2). \]

  \[ (\nu c) \text{compress}(c).c<f_2>.c([p_2]).c(f_3).c([p_3]).c(q_3). \]

  \[ a<f_3>.a([p_3]).a(\text{comb } q_2 \ q_3).0 \]

- **Certificate types**

  \[ q_2 : \Diamond x [\text{agree}(f_2, f_1)] \]

  \[ q_3 : \Diamond c [\text{approx}(f_3, f_2)] \]

  \[ \text{comb } q_2 \ q_3 : \Diamond c \Diamond x [\text{approx}(f_3, f_1)] \]
Certificate combination

Certificate types

\[ q_2 : \Diamond_X \text{agree}(f_2, f_1) \]
\[ q_3 : \Diamond_C \text{approx}(f_3, f_2) \]
\[ \text{comb } q_2 \ q_3 : \Diamond_C \Diamond_X \text{approx}(f_3, f_1) \]

Proof

ida : \text{agree}(f_2, f_1) \rightarrow \text{approx}(f_2, f_1)

tra : \text{approx}(f_3, f_2) \rightarrow \text{approx}(f_2, f_1) \rightarrow \text{approx}(f_3, f_1)

\text{comb } q_2 \ q_3 =
\begin{align*}
\text{let } & \langle [q'_3]:[\text{approx}(f_3, f_2)] \rangle_C = q_3 \ \text{in} \\
& \langle \text{let } \langle [q'_2]:[\text{agree}(f_2, f_1)] \rangle_X = q_2 \ \text{in} \\
& \langle [\text{tra } q'_3 \ (\text{ida } q'_2)]: \_ : \_ \rangle_X : \_ \rangle_C
\end{align*}