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).

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

$\$\tau$ A basic data value of type τ

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

Input

$A \multimap B$ Input a session of type A and continue as B

A (too) simple bank service: $\text{bank}_1 \triangleq \text{string} \multimap (\text{nat} \multimap \mathbf{1})$

Output

$A \otimes B$ Output a session of type A and continue as B

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

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

Persistence or Replication

A session behavior that can be accessed multiple times:

$!A$ Replicated session of type A

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

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

Persistent storage: $\text{store}_1 \triangleq !(\text{file} \multimap !(\text{file} \otimes \mathbf{1}))$

Branch and Choice

$A \mathbin{\&} B$ Offer choice between a session of type A and B

$A \oplus B$ Either a session of type A or B (no choice for client)

Money withdrawal: $\mathsf{wd} \triangleq \mathsf{nat} \multimap ((\mathsf{money} \otimes \mathbf{1}) \oplus (\mathsf{string} \otimes \mathbf{1}))$

A more useful bank: $\mathsf{bank}_4 \triangleq !(\mathsf{string} \multimap (\mathsf{dep} \mathbin{\&} \mathsf{wd}))$

Session Types

$A \multimap B$	Input a session of type A and continue as B
$A \otimes B$	Output a session of type A and continue as B
$A \,\&\, B$	Choice between a session of type A or B
$A \oplus B$	Either a session of type A or B
$!A$	A persistent session of type A
$\mathbf{1}$	Terminated session
$\$\tau$	Base types

Typing Judgment

$$\underbrace{u_1{:}A_1, \ldots, u_m{:}A_m}_{\Gamma}; \underbrace{x_1{:}A_1, \ldots, x_n{:}A_n}_{\Delta} \vdash P :: x{:}A$$

Session Types and Intuitionistic Linear Logic

- Its 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.

Motivation

Limitations

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

$$\text{index}_2 \triangleq !(\text{file} \multimap (\text{file} \otimes \mathbf{1}))$$

  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 τ is not specified.
- We will consider types τ from a (dependent) type theory.
  - Types can depend on terms
  - In practice, types denote *properties*, terms denote *proofs*.
- Generalize idioms $\tau \multimap A$ and $\tau \otimes A$.

Generalizing the Interpretation - Term Input

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

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

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

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: $\text{index}_3 \triangleq !(\forall f{:}\text{file.pdf}(f) \multimap \text{file} \otimes \mathbf{1})$

Even better: $\text{index}_4 \triangleq !(\forall f{:}\text{file.pdf}(f) \multimap \exists g{:}\text{file.pdf}(g) \otimes \mathbf{1})$

Require proof that file is a pdf, supply a proof that output is also a pdf.

- "Glossary" of types:

uid(s)	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

- A bank service (deposits and balance inquiries):

$$\begin{aligned}\text{bank}_5 \triangleq\ & !(\forall s{:}\text{string}.\text{uid}(s) \multimap \\ & (\forall n{:}\text{nat}.\text{deposit}(s,n) \multimap (\text{receipt}(s,n) \otimes \mathbf{1})) \ \& \\ & (\exists m{:}\text{nat}.\text{balance}(s,m) \otimes \mathbf{1}))\end{aligned}$$

After the login, bank offers deposit and balance inquiry services.

- "Glossary" of types:

uid(s)	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):

$$\begin{aligned} \text{atm} \triangleq\ & \forall s{:}\text{string}.\text{uid}(s) \multimap \\ & (\forall n{:}\text{nat}.\text{deposit}(s,n) \multimap \exists m{:}\text{nat}. \\ & \exists p{:}(n-2 \leq m \leq n).\text{receipt}(s,m) \otimes \mathbf{1}) \end{aligned}$$

atm interfaces with bank, charging at most \$2 for a deposit.

Communicating Proofs

Taking Stock

- 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.

Communicating Proofs

One Step Further - Proof Irrelevance

- 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 is a term of type τ that is computationally irrelevant.
- Agree that computationally irrelevant terms will be erased before transmition.
- Typing ensures this can be done consistently.

Rules

$$\frac{\Psi^{\oplus} \vdash M : \tau}{\Psi \vdash [M] : [\tau]}\ []I \qquad \frac{\Psi \vdash M : [\tau] \quad \Psi, x \div \tau \vdash N : \sigma}{\Psi \vdash \textbf{let}\ [x] = M\ \textbf{in}\ N : \sigma}\ []E$$

($\Psi^{\oplus}$ promotes hypotheses $x \div \tau$ to $x : \tau$)

- Typing guarantees that terms $[M]$ can be erased safely.

Using Proof Irrelevance

We mark proofs as computationally irrelevant:

$$\text{index}_4 \triangleq\ !(\forall f{:}\text{file}.\text{pdf}(f) \multimap \exists g{:}\text{file}.\text{pdf}(g) \otimes \mathbf{1})$$
$$\text{index}_5 \triangleq\ !(\forall f{:}\text{file}.[\text{pdf}(f)] \multimap \exists g{:}\text{file}.[\text{pdf}(g)] \otimes \mathbf{1})$$

- A revised atm service:

$$\begin{aligned}\text{atm}_2 \triangleq\ & \forall s{:}\text{string}.\text{uid}(s) \multimap \\ & (\forall n{:}\text{nat}.\text{deposit}(s,n) \multimap \exists m{:}\text{nat}. \\ & \exists p{:}[n-2 \leq m \leq n].\text{receipt}(s,m) \otimes \mathbf{1})\end{aligned}$$

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

- A verifying indexer:

$$\begin{aligned}\text{index}_6 \triangleq\ & !(\forall f{:}\text{file}.[\text{pdf}(f)] \multimap \exists g{:}\text{file}.[\text{pdf}(g)] \otimes \\ & ([\mathit{agree}(f,g)] \otimes \mathbf{1}))\end{aligned}$$

  agree(f, g) if f and g differ only in the index.

Communicating Proofs

Digital Signatures and Affirmation

- 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.
- $M :_K \tau$: Principal K affirms property τ, with evidence M.

Affirmation

$$\frac{\Psi \vdash M : \tau}{\Psi \vdash \langle M : \tau \rangle_K :_K \tau} \text{ (affirms)}$$

$$\frac{\Psi \vdash M :_K \tau}{\Psi \vdash M : \Diamond_K \tau} \Diamond I \quad \frac{\Psi \vdash M : \Diamond_K \tau \quad \Psi, x : \tau \vdash N :_K \sigma}{\Psi \vdash \textbf{let } \langle x : \tau \rangle_K = M \textbf{ in } N :_K \sigma} \Diamond E$$

- 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) $\Diamond_K \tau$.

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

- PDF indexing service, with indexer X:

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

- ATM, identified by principal A:

$$\text{atm}_3 \triangleq \forall s{:}\text{string}.\text{uid}(s) \multimap (\forall n{:}\text{nat}.\text{deposit}(s, n) \multimap \exists m{:}\text{nat}. \exists p{:}\Diamond_A[n - 2 \leq m \leq n].\text{receipt}(s, m) \otimes \mathbf{1})$$

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

- Recall the PDF indexer:

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

- A PDF compression service, with compressor C:

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

- A composite service, indexing and compression:

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

- Affirmation tracks the principals that need to be trusted!

- Affirmation tracks provenance and info. flow:
  - In general, not the case that $\Diamond_K \tau \to \tau$.
  - Needs a form declassification.
- Modelling trust from affirmations:
  - For specific types τ and principals K:

$$\text{trust}_{K,\tau} : \Diamond_K \tau \to \tau$$

  - Implementable by stripping signature.
  - Erased proofs $[\tau]$ cannot generally be recovered:
    - $\not\vdash [\tau] \to \tau$
    - $\not\vdash \Diamond_K [\tau] \to \tau$

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.

Conclusion and Future Work

Assessment

- 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.

Related Work

- “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

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

- Implementation

$$\begin{aligned}&\text{consolidator} = \\ &!\text{ixc}(a).a(f_1).a([p_1]). \\ &\quad (\nu b)\mathit{index}\langle b\rangle.b\langle f_1\rangle.b\langle [p_1]\rangle.b(f_2).b([p_2]).b(q_2). \\ &\quad\quad (\nu c)\text{compress}\langle c\rangle.c\langle f_2\rangle.c\langle [p_2]\rangle.c(f_3).c([p_3]).c(q_3). \\ &\quad\quad\quad a\langle f_3\rangle.a\langle [p_3]\rangle.a\langle \text{comb}\ q_2\ q_3\rangle.\mathbf{0}\end{aligned}$$

- Certificate types

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

Certificate combination

- Certificate types

$$\begin{array}{rcl} q_2 & : & \Diamond_X[\text{agree}(f_2, f_1)] \\ q_3 & : & \Diamond_C[\text{approx}(f_3, f_2)] \\ \text{comb}\ q_2\ q_3 & : & \Diamond_C\Diamond_X[\text{approx}(f_3, f_1)] \end{array}$$

- Proof

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

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

$$\begin{array}{l} \text{comb}\ q_2\ q_3 = \\ \quad \textbf{let}\ \langle [q_3'] : [\text{approx}(f_3, f_2)] \rangle_C = q_3\ \textbf{in} \\ \quad\quad \langle \textbf{let}\ \langle [q_2'] : [\text{agree}(f_2, f_1)] \rangle_X = q_2\ \textbf{in} \\ \quad\quad\quad \langle [\text{tra}\ q_3'\ (\text{ida}\ q_2')] : \_ \rangle_X : \_ \rangle_C \end{array}$$