Dependent Session Types via Intuitionistic Linear Type Theory

Frank Pfenning
[with Luís Caires and Bernardo Toninho]

Department of Computer Science
Carnegie Mellon University

Workshop on Behavioural Types
April 20, 2010 / Lisbon
Outline

1. Introduction
2. Value Types
3. Interface Contracts and Quantification
4. An Extended Example
5. Proof Irrelevance
6. Conclusion
Outline

1. Introduction
2. Value Types
3. Interface Contracts and Quantification
4. An Extended Example
5. Proof Irrelevance
6. Conclusion
Overview

- Session types are mostly “simple” types
  - Emphasis on communication behavior
  - No complex contracts on values
- Exploit logical foundations of session types
  - Proof-theoretic semantics
  - Computation derived from cut reduction
- An analogy
  - Simple types as propositions [Curry-Howard’69]
  - Dependent types for expressive specifications [Martin-Löf’80]
  - Session types as linear propositions
  - Dependent session types for expressive contracts
- Proof irrelevance
  - Bridge between dependent and simple types
  - May erase computationally irrelevant proofs
  - New considerations in distributed settings
Why Curry-Howard?

- Orthogonality of constructs, properties
  - ND/FUN: $\rightarrow$, $\times$, 1, +, 0
  - DILL/SES: $\rightarrow\circ$, $\otimes$, 1, $\oplus$, $\&$, !

- Systematic proof-theoretic foundation
  - ND/FUN: proof reduction gives rise to computation
  - DILL/SES: cut reduction gives rise to computation

- Co-design of computational system with logic for reasoning
  - ND/FUN: Dependent types, inductive types and recursion
  - DILL/SES: Quantification and contracts
How to Read the Judgments

\[
\frac{u_1:A_1, \ldots, u_n:A_n ; x_1:B_1, \ldots, x_k:B_k}{\Gamma, \Delta} \Rightarrow P :: z : C
\]

- Process $P$ provides service $C$ along channel $z$ ...
- ... when composed with processes
  - providing persistent services $A_i$ along $u_i$ and
  - providing (linear) services $B_j$ along $x_j$
Linear Session Type Summary

\[ P :: z : A \rightarrow B \] Input an \( A \) along \( z \) and behave as \( B \)

\[ P :: z : A \otimes B \] Output a new \( x:A \) along \( z \) and behave as \( B \)

\[ P :: z : 1 \] Terminate

\[ P :: z : !A \] Persistently offer \( A \) along \( z \)

\[ P :: z : A \& B \] Offer both \( A \) and \( B \) along \( z \)

\[ P :: z : A \oplus B \] Offer either \( A \) or \( B \) along \( z \)
Outline

1. Introduction
2. Value Types
3. Interface Contracts and Quantification
4. An Extended Example
5. Proof Irrelevance
6. Conclusion
Value Types

- So far, we type only channels
- Add values from an underlying (functional) language
- \[ P :: z : $\tau \] — Provide value of type \( \tau \) along \( z \)
- Examples:

\[
\begin{align*}
P :: z : &\text{nat} \rightarrow \text{nat} \otimes 1 & \text{Increment argument} \\
P :: z : &\text{string} \rightarrow \text{nat} \otimes 1 & \text{Balance inquiry} \\
P :: z : &\text{string} \rightarrow \text{nat} \rightarrow \text{string} \otimes 1 & \text{Deposit with receipt} \\
P :: z : &!(\!(\text{string} \rightarrow \text{nat} \otimes 1) \\
&\& (\!(\text{string} \times \text{nat}) \rightarrow \text{string} \otimes 1)\!)) & \text{A Bank}
\end{align*}
\]
Logical Rules = Typing Rules

- Give right and left rules, as usual
- Aux. judgment $\underbrace{x_1: \tau_1, \ldots, x_n: \tau_n}_\Psi \vdash M : \tau$
- Generalize sequent to $\Psi; \Gamma; \Delta \Rightarrow P :: z : C$
- $\Psi$ is persistent (not the only choice . . .)
- Right rule

$$
\frac{\Psi \vdash M : \tau}{\Psi; \Gamma; \cdot \Rightarrow [x \leftarrow M] :: x : \tau} \quad \text{\$R$}
$$

- Left rule

$$
\frac{\Psi, x: \tau; \Gamma; \Delta \Rightarrow Q :: z : C}{\Psi; \Gamma; \Delta, x: \tau \Rightarrow Q :: z : C} \quad \text{\$L$}
$$
Cut Reduction = Computation

\[
\begin{align*}
\frac{\psi \vdash M : \tau}{\psi; \Gamma; \cdot \rightarrow [x \leftarrow M] :: x : $\tau$} & \quad \text{(R)} \\
\frac{\psi, x:\tau; \Gamma; \Delta \Rightarrow Q(x) :: z : C}{\psi; \Gamma; \Delta, x:$\tau \rightarrow Q(x) :: z : C} & \quad \text{(L)} \\
\frac{\psi; \Gamma; \Delta \Rightarrow (\nu x)([x \leftarrow M] \mid Q(x)) :: z : C}{\psi; \Gamma; \Delta \Rightarrow Q(M) :: z : C} & \quad \text{(Cut)}
\end{align*}
\]

- Reduction \((\nu x)([x \leftarrow M] \mid Q(x)) \rightarrow Q(M)\)
- Requires substitution principle:

\[
\text{If } \psi \vdash M : \tau \text{ and } \psi, x:\tau \vdash J(x) \text{ then } \psi \vdash J(M).
\]
Examples

- Increment

\[
\text{inc} \quad :: \quad z : \text{nat} \rightarrow \text{nat} \otimes 1 \\
= \quad z(n). (\nu x) z\langle x \rangle . ([x \leftarrow n + 1] \mid 0)
\]

- Balance inquiry, with bal : string \rightarrow nat

\[
\text{inq} \quad :: \quad z : \text{string} \rightarrow \text{nat} \otimes 1 \\
= \quad z(s). (\nu x) z\langle x \rangle . ([x \leftarrow \text{bal}(s)] \mid 0)
\]

- Deposit with receipt, with rct : string \times \text{nat} \rightarrow \text{string}

\[
\text{dep} \quad :: \quad z : \text{string} \rightarrow \text{nat} \rightarrow \text{string} \otimes 1 \\
= \quad z(s). z(n). (\nu x) z\langle x \rangle . ([x \leftarrow \text{rct}(s, n)] \mid 0)
\]
Outline

1. Introduction

2. Value Types

3. Interface Contracts and Quantification

4. An Extended Example

5. Proof Irrelevance

6. Conclusion
Interface Contracts

- Types so far are imprecise
- Interface contract examples
  - Increment returns greater number
  - Increment returns $n + 1$
  - Balance inquiry for authenticated user receives a signed statement
  - Deposit of authenticated user receives a signed receipt
  - ATM deducts a fee of at most $2$ per transaction

Solution

- Quantification in session types
- Dependent types in (functional) substrait

Purely logical!

- Follow the proof theory . . .
Quantification

$P :: z : \forall x : \tau. A(x)$  
Input an $M : \tau$ along $z$ and behave as $A(M)$

$P :: z : \exists x : \tau. A(x)$  
Output an $M : \tau$ along $z$ and behave as $A(M)$

- Increment returns larger result

  $P :: z : \forall n : \text{nat}. \exists n' : \text{nat}. (n' > n) \otimes 1$

- Increment increments

  $P :: z : \forall n : \text{nat}. \exists n' : \text{nat}. (n' = n + 1) \otimes 1$

- Balance inquiry for auth’d user receives a signed statement

  $P :: z : \forall s : \text{string}. \Auth(s) \rightarrow \exists n : \text{nat}. \Bal(s, n) \otimes 1$
Logical Rules = Typing Rules

- \( P :: z : \forall x : \tau. A(x) \)  
  Input an \( M : \tau \) along \( z \) and behave as \( A(M) \)

- Give right and left rules, as always

- Right rule

  \[
  \frac{\psi, y : \tau; \Gamma; \Delta \Rightarrow P(y) :: x : A(y)}{
  \psi; \Gamma; \Delta \Rightarrow x(y). P(y) :: x : \forall y : \tau. A(y)}
  \]
  \( \forall R \)

- Left rule

  \[
  \frac{\psi \vdash M : \tau \quad \psi; \Delta, x : A(M) \Rightarrow Q :: z : C}{
  \psi; \Gamma; \Delta, x : \forall y : \tau. A(y) \Rightarrow x(M) . Q :: z : C}
  \]
  \( \forall L \)
Cut Reduction = Computation

\[
\begin{align*}
\Rightarrow P(y) :: x : A(y) & \quad \forall R^y \\
\Rightarrow x(y). P(y) :: x : \forall y : \tau. A(y) & \quad \forall L \\
\Rightarrow (\nu x)(x(y). P(y) | x\langle M\rangle. Q) :: z : C & \quad \text{Cut}
\end{align*}
\]

\[
\begin{align*}
\Rightarrow P(M) :: x : A(M) & \quad x : A(M) \Rightarrow Q :: z : C \\
\Rightarrow (\nu x)(P(M) | Q) :: z : C & \quad \text{Cut}
\end{align*}
\]

- (omitted contexts)
- Reduction rule extracted

\[
x(y). P(y) | x\langle M\rangle. Q \rightarrow P(M) | Q
\]

- Already known, except passing values, not channels
Existential Quantification

- \( P :: z : \exists x : \tau. A(x) \)  
  Output an \( M : \tau \) along \( z \) and behave as \( A(M) \)
- Existential quantification is dual to universal quantification
- Right rule

\[
\psi \vdash M : \tau \\
\psi ; \Gamma ; \Delta \Rightarrow P :: x : A(M) \\
\psi ; \Gamma ; \Delta \Rightarrow x(M) . P :: x : \exists y : \tau. A(y)
\]

- Left rule

\[
\psi \vdash M : \tau \\
\psi , y : \tau ; \Gamma , \Delta , x : A(y) \Rightarrow Q(y) :: z : C \\
\psi ; \Gamma , \Delta , x : \exists y : \tau. A(y) \Rightarrow x(y) . Q(y) :: z : C
\]

- No new reduction

\[
x(M) . P | x(y) . Q(y) \longrightarrow P | Q(M)
\]
Example Revisited: Increment

- Types such as \( m > n \) or \( m = n \) are inhabited by proofs; this applies to full functional specifications in type theory.
- Use standard \( \Pi x: \tau . \sigma \) and \( \Sigma x: \tau . \sigma \) from type theory in functional substrate.
- Increment returns a larger result, using \( \text{gt}_1 : \Pi k : \text{nat}. \ k + 1 > k \):

\[
\text{inc} :: z : \forall n : \text{nat}. \ \exists n' : \text{nat}. \ (n' > n) \otimes 1 \\
= \ z(n). \ z\langle n+1 \rangle. (\nu x)([x \leftarrow \text{gt}_1(n)] \mid 0)
\]

- Increment increments, using \( \text{refl} : \Pi k : \text{nat}. \ k = k \):

\[
\text{inc} :: z : \forall n : \text{nat}. \ \exists n' : \text{nat}. \ (n' = n + 1) \otimes 1 \\
= \ z(n). \ z\langle n+1 \rangle. (\nu x)([x \leftarrow \text{refl(n + 1)}] \mid 0)
\]
Example Revisited: Balance Inquiry

- Balance inquiry for auth’d user receives a signed statement

\[ P :: z : \forall s: \text{string}. \ auth(s) \implies \exists n: \text{nat}. \ bal(s, n) \otimes 1 \]

- Types such as auth(s) or bal(s, n) are inhabited by cryptographically signed certificates, or proofs in an authorization logic constructed from them

- Process, with bl : \( \prod s: \text{string. } \Sigma n: \text{nat. } \text{bal}(s, n) \)

\[
\text{inq :: } z : \forall s: \text{string. } \text{auth}(s) \implies \exists n: \text{nat. } \text{bal}(s, n) \otimes 1 \\
\overset{=} \text{z(s) \cdot z(a) \cdot z(\pi_1(\text{bl}(s))) \cdot (\nu x)((x \leftarrow \pi_2(\text{bl}(s))) \mid 0)}
\]
Outline

1. Introduction
2. Value Types
3. Interface Contracts and Quantification
4. An Extended Example
5. Proof Irrelevance
6. Conclusion
Example: An ATM

- Mediate between client and bank
- Don’t need $\otimes 1$ to terminate session
- $\textit{BANK}$ provides deposit for any client and provides signed receipt

\[
\textit{Bank} = !\forall s:\text{string}. \forall n:\text{nat}. \exists r:\text{string}. \$\text{receipt}(s, n, r)
\]

\[
\implies \textit{BANK} :: b^* : \textit{Bank}
\]

- $\textit{ATM}$ provides deposit for authenticated client and provides signed receipt. It may deduct at most $2$.

\[
\textit{Atm} = !\forall s:\text{string}. \$\text{auth}(s) \rightarrow \forall n:\text{nat}.
\rightarrow \exists n' : \text{nat}. \$(n' \geq n - 2) \otimes \exists r:\text{string}. \$\text{receipt}(s, n', r)
\]

\[
b^* : \textit{Bank} \implies \textit{ATM} :: a^* : \textit{Atm}
\]

- $\textit{CLIENT}$ uses $\textit{ATM}$

\[
a^* : \textit{Atm} \implies \textit{CLIENT} :: _ : 1
\]
Cut as Composition

We compose BANK and ATM using cut

\[
\cdot \implies BANK :: b^* : Bank \quad b^* : Bank \implies ATM :: a^* : Atm
\]

\[
\cdot \implies (\nu b^*)(BANK \mid ATM) :: a^* : Atm
\]

We compose result and CLIENT using cut

\[
\cdot \implies (\nu b^*)(BANK \mid ATM) :: a^* : Atm \quad a^* : Atm \implies CLIENT :: _ : 1
\]

\[
\cdot \implies (\nu a^*)(((\nu b^*)(BANK \mid ATM)) \mid CLIENT) :: _ : 1
\]

Composition in the other order is structurally congruent

\[
(\nu a^*)(\nu b^*)(BANK \mid ATM \mid CLIENT)
\]

BANK provides \( b^* \), ATM uses \( b^* \) and provides \( a^* \), CLIENT uses \( a^* \)
Implementing ATM

- Recall

\[ Bank = \forall s:\text{string}. \forall n:\text{nat}. \exists r:\text{string}. \$\text{receipt}(s, n, r) \]

\[ Atm = \forall s:\text{string}. \$\text{auth}(s) \rightarrow \forall n:\text{nat}. \]
\[ \rightarrow \exists n':\text{nat}. $(n' \geq n - 2) \otimes \exists r:\text{string}. \$\text{receipt}(s, n', r) \]

\[ b^* : Bank \rightarrow\rightarrow ATM :: a^* : Atm \]

- An implementation, with \( \text{ge}_1 : \Pi k:\text{nat}. k + 1 \geq k \), only \( b^* \) free

\[ ATM = !a^*(a). a(s). a(cert). a(n). \]
\[ (\nu b)(b^*\langle b\rangle. b\langle s\rangle. b\langle n - 1\rangle). \]
\[ b(r). b(rct). \]
\[ a\langle n - 1\rangle. (\nu x)([x \leftarrow \text{ge}_1(n - 2)] | \]
\[ a\langle r\rangle. [a \leftarrow rct]) \]
1. Introduction
2. Value Types
3. Interface Contracts and Quantification
4. An Extended Example
5. Proof Irrelevance
6. Conclusion
Proof Irrelevance

- Sometimes proofs are a burden
  - Can be decided effectively (e.g., increment)
  - Partner can be trusted (e.g., authentication of receipt)
  - Computation is concurrent, but not distributed

- May erase if not computationally relevant
  - Must verify relevant computation does not depend on them
  - Can be checked effectively in a type system

- New type $[\tau]$, computationally irrelevant terms of type $\tau$
- Defined by introduction and elimination rules [Pf.’08]
- May decide to erase or not
  - Progress and preservation hold in either case
Irrelevance as a Modality

- Example: We check the fee ourselves and trust the ATM/Bank

\[
Atm = !\forall s:\text{string}. \ $auth(s) \rightarrow \forall n:\text{nat}.
\]
\[
\rightarrow \exists n':\text{nat}. \ [n' \geq n - 2] \otimes \exists r:\text{string}. \ [\text{receipt}(s, n', r)]
\]

- Example: We don’t want a receipt at all

\[
Atm = !\forall s:\text{string}. \ $auth(s) \rightarrow \forall n:\text{nat}.
\]
\[
\rightarrow \exists n':\text{nat}. \ [n' \geq n - 2] \otimes \exists r:\text{string}. \ [\text{receipt}(s, n', r)]
\]

- Statically, evidence \([M] : [\tau]\) must be provided or inferred
- At run time, \([\ ] : [\tau]\) is sufficient; erase \([\tau]\) to \([\ ]\)
- We can further optimize using erased type isomorphisms, e.g.

\[
[\ ] \times \tau \simeq \tau \simeq [\ ] \times \tau \quad \$[\ ] \otimes A \simeq A \simeq A \otimes \$[\ ]
\]
\[
\Sigma x:[\ ]. \ \sigma \simeq \sigma \quad \exists x:[\ ]. \ A \simeq A
\]
\[
[\ ] \rightarrow \tau \simeq \tau \quad \$[\ ] \rightarrow A \simeq A
\]
Outline

1. Introduction
2. Value Types
3. Interface Contracts and Quantification
4. An Extended Example
5. Proof Irrelevance
6. Conclusion
Variations and Extensions

- Functional substrait is not necessary . . .
  - Can derive concurrent evaluation strategies for functional language
    1. Embed functional language in linear $\lambda$-calculus (std.)
    2. Embed linear $\lambda$-calculus in linear sequent calculus (std.)
      - Result is well-typed in session types

- . . . but good design
  - Separation of concerns

- Inductive and co-inductive types mix with linearity [Baelde’08]
  - Cut reduction (= computation) straightforward unrolling
  - Termination more difficult or does not hold
Ongoing and Future Work

- Observational equivalence as proof conversion
- Irrelevant sessions (speculative)
  - Interaction of linearity and irrelevance [Ley-Wild&Pf.’07]
- Towards a programming language (speculative)
  - Monadic encapsulation of session types?
  - Connection to ML5 (sequential, distributed)? [Murphy’08]
- Towards multiparty session/conversation types (speculative)
  - (Kripke) worlds as conversations?
  - Introducing modalities or hybrid logic formulation
Summary

- Session types as intuitionistic linear propositions:

  \[ A \rightarrow B \] input \hspace{1cm} \forall x: \tau. A(x) \hspace{1cm} \text{value/proof input}
  
  \[ A \otimes B \] (bound) output \hspace{1cm} \exists x: \tau. A(x) \hspace{1cm} \text{value/proof output}
  
  \[ 1 \] inaction \hspace{1cm} \tau \hspace{1cm} \text{value/proof}
  
  \[ A \] replication \hspace{1cm} [\tau] \hspace{1cm} \text{irrelevant term}
  
  \[ A \& B \] external choice \hspace{1cm} A \oplus B \hspace{1cm} \text{internal choice}

- Dependent sessions types via quantification
  - Can express value and proof passing
  - Adherence to expressive logical contracts
  - Satisfies progress and preservation

- Overhead reduction via proof irrelevance
  - Selective hiding based on decidability or trust
  - Avoiding communication by applying type isomorphisms
Soap Box

- Co-design of terms, types, proofs!
  - Constructs can be understood in isolation
  - Reasoning principles built in, not grafted on
  - Path towards extensibility (quantifiers, dependent types)
  - Computation rules as proof reductions

- Draw upon rich intensional concepts in logic!
  - Linearity and sharing — how resources are used
  - Order — how resources are connected
  - Necessity — everwhere and always
  - Possibility — somewhere and sometimes
  - Knowledge — information (flow)
  - Linear knowledge — possession
  - Affirmation — authorization
  - Linear affirmation — use-once authorization
  - Irrelevance — optimizing computation and communication

- Why start from scratch every time?