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

Outline

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: $\multimap$, $\otimes$, $\mathbf{1}$, $\oplus$, $\&$, !
- 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

$$\underbrace{u_1{:}A_1, \ldots, u_n{:}A_n}_{\Gamma} \; ; \; \underbrace{x_1{:}B_1, \ldots, x_k{:}B_k}_{\Delta} \Longrightarrow P :: z : C$$

- 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

$P :: z : A \multimap B$	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
$P :: z : \mathbf{1}$	Terminate
$P :: z : {!A}$	Persistently offer A along z
$P :: z : A \mathbin{\&} B$	Offer both A and B along z
$P :: z : A \oplus B$	Offer either A or B along z

Outline

Value Types

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

$P :: z : \$\mathrm{nat} \multimap \$\mathrm{nat} \otimes \mathbf{1}$	Increment argument
$P :: z : \$\mathrm{string} \multimap \$\mathrm{nat} \otimes \mathbf{1}$	Balance inquiry
$P :: z : \$\mathrm{string} \multimap \$\mathrm{nat} \multimap \$\mathrm{string} \otimes \mathbf{1}$	Deposit with receipt
$P :: z : !((\$\mathrm{string} \multimap \$\mathrm{nat} \otimes \mathbf{1})$	
$\quad \& \, (\$(\mathrm{string} \times \mathrm{nat}) \multimap \$\mathrm{string} \otimes \mathbf{1}))$	A Bank

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 $\Psi; \Gamma; \Delta \Longrightarrow P :: z : C$
- Ψ is persistent (not the only choice ...)
- Right rule

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

- Left rule

$$\frac{\Psi, x{:}\tau; \Gamma; \Delta \Longrightarrow Q :: z : C}{\Psi; \Gamma; \Delta, x{:}\$\tau \Longrightarrow Q :: z : C}\ \$L$$

Cut Reduction = Computation

$$\dfrac{\dfrac{\Psi \vdash M : \tau}{\Psi; \Gamma; \cdot \Longrightarrow [x \leftarrow M] :: x : \$\tau}\ \$R \qquad \dfrac{\Psi, x{:}\tau; \Gamma; \Delta \Longrightarrow Q(x) :: z : C}{\Psi; \Gamma; \Delta, x{:}\$\tau \Longrightarrow Q(x) :: z : C}\ \$L}{\Psi; \Gamma; \Delta \Longrightarrow (\nu x)([x \leftarrow M] \mid Q(x)) :: z : C}\ \mathit{Cut}$$

$$\longrightarrow \quad \Psi; \Gamma; \Delta \Longrightarrow Q(M) :: z : C$$

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

  If $\Psi \vdash M : \tau$ *and* $\Psi, x{:}\tau \vdash J(x)$ *then* $\Psi \vdash J(M)$.

Examples

- Increment

$$\begin{array}{rcl} \mathsf{inc} & :: & z : \$\mathsf{nat} \multimap \$\mathsf{nat} \otimes \mathbf{1} \\ & = & z(n).\,(\nu x)\, z\langle x\rangle.\,([x \leftarrow n+1] \mid \mathbf{0}) \end{array}$$

- Balance inquiry, with bal : string → nat

$$\begin{array}{rcl} \mathsf{inq} & :: & z : \$\mathsf{string} \multimap \$nat \otimes \mathbf{1} \\ & = & z(s).\,(\nu x)\, z\langle x\rangle.\,([x \leftarrow \mathsf{bal}(s)] \mid \mathbf{0}) \end{array}$$

- Deposit with receipt, with rct : string × nat → string

$$\begin{array}{rcl} \mathsf{dep} & :: & z : \$\mathsf{string} \multimap \$\mathsf{nat} \multimap \$\mathsf{string} \otimes 1 \\ & = & z(s).\, z(n).\,(\nu x)\, z\langle x\rangle.\,([x \leftarrow \mathsf{rct}(s, n)] \mid \mathbf{0}) \end{array}$$

Outline

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

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

- Increment increments

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

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

$$P :: z : \forall s{:}\mathrm{string}.\, \$\mathrm{auth}(s) \multimap \exists n{:}\mathrm{nat}.\, \$\mathrm{bal}(s, n) \otimes \mathbf{1}$$

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

$$\frac{\Psi, y{:}\tau; \Gamma; \Delta \Longrightarrow P(y) :: x : A(y)}{\Psi; \Gamma; \Delta \Longrightarrow x(y).\, P(y) :: x : \forall y{:}\tau.\, A(y)}\ \forall R$$

- Left rule

$$\frac{\Psi \vdash M : \tau \quad \Psi; \Gamma; \Delta, x{:}A(M) \Longrightarrow Q :: z : C}{\Psi; \Gamma; \Delta, x{:}\forall y{:}\tau.\, A(y) \Longrightarrow x\langle M\rangle.\, Q :: z : C}\ \forall L$$

Cut Reduction = Computation

$$
\dfrac{\dfrac{\Longrightarrow P(y) :: x : A(y)}{\Longrightarrow x(y).\,P(y) :: x : \forall y{:}\tau.\,A(y)}\ \forall R^{y} \qquad \dfrac{M : \tau \quad x{:}A(M) \Longrightarrow Q :: z : C}{x{:}\forall y{:}\tau.\,A(y) \Longrightarrow x\langle M\rangle.\,Q :: z : C}\ \forall L}{\Longrightarrow (\nu x)(x(y).\,P(y) \mid x\langle M\rangle.\,Q) :: z : C}\ \mathit{Cut}
$$

$$
\longrightarrow \qquad \dfrac{\Longrightarrow P(M) :: x : A(M) \quad x{:}A(M) \Longrightarrow Q :: z : C}{\Longrightarrow (\nu x)(P(M) \mid Q) :: z : C}\ \mathit{Cut}
$$

- (omitted contexts)
- Reduction rule extracted

$$
x(y).\,P(y) \mid x\langle M\rangle.\,Q \longrightarrow P(M) \mid Q
$$

- 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

$$\frac{\Psi \vdash M : \tau \quad \Psi; \Gamma; \Delta \Longrightarrow P :: x : A(M)}{\Psi; \Gamma; \Delta \Longrightarrow x\langle M\rangle.\, P :: x : \exists y{:}\tau.\, A(y)}\ \exists R$$

- Left rule

$$\frac{\Psi \vdash M : \tau \quad \Psi, y{:}\tau; \Gamma; \Delta, x{:}A(y) \Longrightarrow Q(y) :: z : C}{\Psi; \Gamma; \Delta, x{:}\exists y{:}\tau.\, A(y) \Longrightarrow x(y).\, Q(y) :: z : C}\ \exists L$$

- No new reduction

$$x\langle M\rangle.\, P \mid x(y).\, Q(y) \longrightarrow P \mid Q(M)$$

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 substrait
- Increment returns a larger result, using $\mathsf{gt}_1 : \Pi k{:}\mathsf{nat}.\, k + 1 > k$

$$\begin{array}{lcl} \mathsf{inc} & :: & z : \forall n{:}\mathsf{nat}.\, \exists n'{:}\mathsf{nat}.\, \$(n' > n) \otimes \mathbf{1} \\ & = & z(n).\, z\langle n+1\rangle.\, (\nu x)([x \leftarrow \mathsf{gt}_1(n)] \mid \mathbf{0}) \end{array}$$

- Increment increments, using $\mathsf{refl} : \Pi k{:}\mathsf{nat}.\, k = k$

$$\begin{array}{lcl} \mathsf{inc} & :: & z : \forall n{:}\mathsf{nat}.\, \exists n'{:}\mathsf{nat}.\, \$(n' = n+1) \otimes \mathbf{1} \\ & = & z(n).\, z\langle n+1\rangle.\, (\nu x)([x \leftarrow \mathsf{refl}(n+1)] \mid \mathbf{0}) \end{array}$$

Example Revisited: Balance Inquiry

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

$$P :: z : \forall s{:}\mathrm{string}.\, \$\mathrm{auth}(s) \multimap \exists n{:}\mathrm{nat}.\, \$\mathrm{bal}(s, n) \otimes \mathbf{1}$$

- Types such as $\mathrm{auth}(s)$ or $\mathrm{bal}(s, n)$ are inhabited by cryptographically signed certificates, or proofs in an authorization logic constructed from them
- Process, with $\mathrm{bl} : \Pi s{:}\mathrm{string}.\, \Sigma n{:}\mathrm{nat}.\, \mathrm{bal}(s, n)$

$$\begin{array}{lcl} \mathrm{inq} & :: & z : \forall s{:}\mathrm{string}.\, \$\mathrm{auth}(s) \multimap \exists n{:}\mathrm{nat}.\, \$\mathrm{bal}(s, n) \otimes \mathbf{1} \\ & = & z(s).\, z(a).\, z\langle \pi_1(\mathrm{bl}(s)) \rangle.\, (\nu x)([x \leftarrow \pi_2(\mathrm{bl}(s))] \mid \mathbf{0}) \end{array}$$

Outline

Example: An ATM

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

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

$$\cdot \Longrightarrow BANK :: b^* : Bank$$

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

$$\begin{aligned} Atm = {} & !\forall s{:}\mathrm{string}.\, \$\mathrm{auth}(s) \multimap \forall n{:}\mathrm{nat}. \\ & \multimap \exists n'{:}\mathrm{nat}.\, \$(n' \geq n - 2) \otimes \exists r{:}\mathrm{string}.\, \$\mathrm{receipt}(s, n', r) \end{aligned}$$

$$b^* : Bank \Longrightarrow ATM :: a^* : Atm$$

- *CLIENT* uses *ATM*

$$a^* : Atm \Longrightarrow CLIENT :: \_ : \mathbf{1}$$

Cut as Composition

- We compose BANK and ATM using cut

$$\frac{\cdot \Longrightarrow BANK :: b^* : Bank \quad b^* : Bank \Longrightarrow ATM :: a^* : Atm}{\cdot \Longrightarrow (\nu b^*)(BANK \mid ATM) :: a^* : Atm}\ Cut$$

- We compose result and CLIENT using cut

$$\frac{\cdot \Longrightarrow (\nu b^*)(BANK \mid ATM) :: a^* : Atm \quad a^* : Atm \Longrightarrow CLIENT :: \_ : \mathbf{1}}{\cdot \Longrightarrow (\nu a^*)(((\nu b^*)(BANK \mid ATM)) \mid CLIENT) :: \_ : \mathbf{1}}\ Cut$$

- 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

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

$$\begin{aligned} Atm = {} & !\forall s{:}\mathrm{string}.\, \$\mathrm{auth}(s) \multimap \forall n{:}\mathrm{nat}. \\ & \quad \multimap \exists n'{:}\mathrm{nat}.\, \$(n' \geq n - 2) \otimes \exists r{:}\mathrm{string}.\, \$\mathrm{receipt}(s, n', r) \end{aligned}$$

$$b^* : Bank \Longrightarrow ATM :: a^* : Atm$$

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

$$\begin{aligned} ATM \quad = \quad & !a^*(a).\, a(s).\, a(cert).\, a(n). \\ & \quad (\nu b)(b^*\langle b\rangle.\, b\langle s\rangle.\, b\langle n - 1\rangle. \\ & \qquad b(r).\, b(rct). \\ & \qquad a\langle n - 1\rangle.\, (\nu x)([x \leftarrow \mathrm{ge}_1(n - 2)] \mid \\ & \qquad\qquad a\langle r\rangle.\, [a \leftarrow rct])) \end{aligned}$$

Outline

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 τ
- 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

$$
\begin{aligned}
Atm = {} & !\forall s{:}\mathsf{string}.\,\$\mathsf{auth}(s) \multimap \forall n{:}\mathsf{nat}. \\
& \multimap \exists n'{:}\mathsf{nat}.\,\$[n' \geq n-2] \otimes \exists r{:}\mathsf{string}.\,\$[\mathsf{receipt}(s, n', r)]
\end{aligned}
$$

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

$$
\begin{aligned}
Atm = {} & !\forall s{:}\mathsf{string}.\,\$\mathsf{auth}(s) \multimap \forall n{:}\mathsf{nat}. \\
& \multimap \exists n'{:}\mathsf{nat}.\,\$[n' \geq n-2] \otimes \exists r{:}[\mathsf{string}].\,\$[\mathsf{receipt}(s, n', r)]
\end{aligned}
$$

- 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.

$$
\begin{array}{ll}
[\,] \times \tau \simeq \tau \simeq [\,] \times \tau & \$[\,] \otimes A \simeq A \simeq A \otimes \$[\,] \\
\Sigma x{:}[\,].\,\sigma \simeq \sigma & \exists x{:}[\,].\,A \simeq A \\
[\,] \to \tau \simeq \tau & \$[\,] \multimap A \simeq A
\end{array}
$$

Outline

Variations and Extensions

- Functional substrait is not necessary ...
  - Can derive concurrent evaluation strategies for functional language
  1. Embed functional language in linear λ-calculus (std.)
  2. Embed linear λ-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:

$A \multimap B$	input	$\forall x{:}\tau.\, A(x)$	value/proof input
$A \otimes B$	(bound) output	$\exists x{:}\tau.\, A(x)$	value/proof output
$\mathbf{1}$	inaction	$\$\tau$	value/proof
$!A$	replication	$[\tau]$	irrelevant term
$A \mathbin{\&} B$	external coice	$A \oplus B$	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?