Focusing on Binding and Computation

Robert Harper

Carnegie Mellon University

TLCA/RTA
July 2009

Thanks

Thanks especially to Daniel R. Licata and Noam Zeilberger, my collaborators on much of this work.

Thanks also to Frank Pfenning, Steve Awodey, Peter Lumsdaine, and Lars Birkedal.

And thanks to the TLCA and RTA organizers for the invitation!

Focused Proofs

Andreoli: focusing proof search for classical linear logic.

- Refinement of cut-free proofs for more effective proof search.
- Implementations and generalizations by Pfenning, Miller, Chaudhuri, and others.

Two key ideas:

- Invertibility: control "don't care" indeterminacy.
- Focusing: control "don't know" indeterminacy.

Inversion

A rule is invertible if the premises are derivable from the conclusion:

$$\text{Left}: \quad \frac{\Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C} \qquad \text{Right}: \quad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}$$

An inversion step composes invertible rules:

$$\text{Left}: \quad \frac{\Gamma, A \vdash D \quad \Gamma, B \vdash D \quad \Gamma, C \vdash D}{\Gamma, A \vee (B \vee C) \vdash D} \qquad \text{Right}: \quad \frac{\Gamma, A, B \vdash C}{\Gamma \vdash A \supset (B \supset C)}$$

Focus

Non-invertible rules involve choices:

$$\text{Left}: \quad \frac{\Gamma \vdash A \quad \Gamma, B \vdash C}{\Gamma, A \supset B \vdash C} \qquad \text{Right}: \quad \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}$$

A focusing step composes choices:

$$\text{Left}: \quad \frac{\Gamma \vdash A \quad \Gamma \vdash B \quad \Gamma, C \vdash D}{\Gamma, A \supset (B \supset C) \vdash D} \qquad \text{Right}: \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee (B \vee C)}$$

Polarities

In linear logic the connectives may be classified by polarity:

- Positive: left invertible, right focus. $\otimes$, $\oplus$, $\mathbf{1}$, $\mathbf{0}$.
- Negative: right invertible, left focus. $\multimap$, $\&$, $\top$, $\parr$.

Girard: distinguish positive and negative connectives a priori.

- Positive = verificationist = eager = inductive.
- Negative = pragmatist = lazy = coinductive.

Zeilberger: provides link to type systems via pattern matching.

Polarities

Positive types: defined by introduction.

- Right focus: choose a (compound) value of a type.
- Left inversion: match all possible values.

Negative types: defined by elimination.

- Left focus: choose a (compound) experiment for a type.
- Right inversion: match all possible experiments.

Cut elimination establishes safety via exhaustiveness of matching.

Positive Types

Positive sum: $A_1^+ \oplus A_2^+$.

- Introduce by choosing a value:

$$\mathsf{inl} \circ v_1^+$$
$$\mathsf{inr} \circ v_2^+$$

- Eliminate by matching all values:

$$\mathsf{inl} \circ v_1^+ \longmapsto m_1$$
$$\mathsf{inr} \circ v_2^+ \longmapsto m_2$$

Negative Types

Negative product: $A_1^- \& A_2^-$.

- Eliminate by choosing an experiment:

$$\begin{aligned} \mathsf{fst};\, k_1^- \\ \mathsf{snd};\, k_2^- \end{aligned}$$

- Introduce by matching all experiments:

$$\begin{aligned} \mathsf{fst};\, k_1^- &\mapsto m_1 \\ \mathsf{snd};\, k_2^- &\mapsto m_2 \end{aligned}$$

Polarities

Positive types:

$$A^+ ::= \mathbf{1} \mid \mathbf{0} \mid A_1^+ \otimes A_2^+ \mid A_1^+ \oplus A_2^+ \mid \downarrow A^-$$

Negative types:

$$A^- ::= A_1^- \& A_2^- \mid A_1^+ \to A_2^- \mid \uparrow A^+$$

Shift operators intermix positive and negative:

- $\downarrow A^-$: inclusion of negative into positive.
- $\uparrow A^+$: suspension of positive computation.

Focusing

Positive fragment:

$$\text{Right Focus} = \text{Value} \qquad \Gamma \vdash \quad A^+$$

$$\text{Left Inversion} = \text{Match} \qquad \Gamma \vdash \quad A^+ > \gamma$$

Negative fragment:

$$\text{Left Focus} = \text{Experiment} \qquad \Gamma \vdash \quad A^- > \gamma$$

$$\text{Right Inversion} = \text{Response} \qquad \Gamma \vdash \quad A^-$$

Neutral (computation) fragment (with result γ):

$$\Gamma \vdash_0 \quad \gamma$$

Focusing

Positive fragment:

$$\text{Right Focus} = \text{Value} \qquad \Gamma \vdash v^+ : A^+$$

$$\text{Left Inversion} = \text{Match} \qquad \Gamma \vdash k^+ : A^+ > \gamma$$

Negative fragment:

$$\text{Left Focus} = \text{Experiment} \qquad \Gamma \vdash \quad A^- > \gamma$$

$$\text{Right Inversion} = \text{Response} \qquad \Gamma \vdash \quad A^-$$

Neutral (computation) fragment (with result γ):

$$\Gamma \vdash_0 \quad \gamma$$

Focusing

Positive fragment:

$$\text{Right Focus} = \text{Value} \qquad \Gamma \vdash v^+ : A^+$$

$$\text{Left Inversion} = \text{Match} \qquad \Gamma \vdash k^+ : A^+ > \gamma$$

Negative fragment:

$$\text{Left Focus} = \text{Experiment} \qquad \Gamma \vdash k^- : A^- > \gamma$$

$$\text{Right Inversion} = \text{Response} \qquad \Gamma \vdash v^- : A^-$$

Neutral (computation) fragment (with result γ):

$$\Gamma \vdash_0 \qquad \gamma$$

Focusing

Positive fragment:

Right Focus = Value
$\Gamma \vdash v^+ : A^+$

Left Inversion = Match
$\Gamma \vdash k^+ : A^+ > \gamma$

Negative fragment:

Left Focus = Experiment
$\Gamma \vdash k^- : A^- > \gamma$

Right Inversion = Response
$\Gamma \vdash v^- : A^-$

Neutral (computation) fragment (with result γ):

$$\Gamma \vdash_0 m : \gamma$$

Focusing

Positive disjunction (derivable):

Right Focus

$$\frac{\Gamma \vdash \quad A_1^+}{\Gamma \vdash \quad A_1^+ \oplus A_2^+}$$

$$\frac{\Gamma \vdash \quad A_2^+}{\Gamma \vdash \quad A_1^+ \oplus A_2^+}$$

Left Inversion

$$\frac{\Gamma \vdash \quad A_1^+ > \gamma \quad \Gamma \vdash \quad A_2^+ > \gamma}{\Gamma \vdash \quad A_1^+ \oplus A_2^+ > \gamma}$$

Focusing

Positive disjunction (derivable):

Right Focus

$$\frac{\Gamma \vdash v^+ : A_1^+}{\Gamma \vdash \mathsf{inl} \circ v^+ : A_1^+ \oplus A_2^+}$$

$$\frac{\Gamma \vdash v^+ : A_2^+}{\Gamma \vdash \mathsf{inr} \circ v^+ : A_1^+ \oplus A_2^+}$$

Left Inversion

$$\frac{\Gamma \vdash k_1 : A_1^+ > \gamma \quad \Gamma \vdash k_2 : A_2^+ > \gamma}{\Gamma \vdash \{\, \mathsf{inl} \mapsto k_1 \mid \mathsf{inr} \mapsto k_2 \,\} : A_1^+ \oplus A_2^+ > \gamma}$$

Higher-Order Focusing

Choices and patterns are maximal.

- Short-cut pattern matches are definable (identity lemma).
- All variables are of negative type.

A positive value v^+ may be decomposed into $p^+[\sigma]$ where

- p^+ is a positive pattern, which can be matched, and
- σ is a negative substitution for variables, which are opaque.

Example: $\mathbf{1} \oplus (\mathbf{1} \oplus \downarrow A^-)$.

- Patterns: $\mathsf{inl} \circ \langle\rangle$, $\mathsf{inr} \circ \mathsf{inl} \circ \langle\rangle$, $\mathsf{inr} \circ \mathsf{inr} \circ x$.
- Values: $\mathsf{inl} \circ \langle\rangle$, $\mathsf{inr} \circ \mathsf{inl} \circ \langle\rangle$, $(\mathsf{inr} \circ \mathsf{inr} \circ x)[v^-/x]$.

Higher-Order Focusing

Patterns are the fulcrum of the type theory:

$$\Delta \Vdash \quad A^+ \qquad\qquad \Delta \Vdash \quad A^- > \gamma$$

Example: positive disjunction patterns.

$$\frac{\Delta \Vdash \quad A_1^+}{\Delta \Vdash \quad A_1^+ \oplus A_2^+} \qquad \frac{\Delta \Vdash \quad A_2^+}{\Delta \Vdash \quad A_1^+ \oplus A_2^+}$$

Example: negative function patterns.

$$\frac{\Delta \Vdash \quad A_1^+ \quad \Delta \Vdash \quad A_2 > \gamma}{\Delta \Vdash \quad (A_1^+ \to A_2^-) > \gamma}$$

Higher-Order Focusing

Patterns are the fulcrum of the type theory:

$$\Delta \Vdash p^+ : A^+ \qquad\qquad \Delta \Vdash p^- : A^- > \gamma$$

Example: positive disjunction patterns.

$$\frac{\Delta \Vdash \quad A_1^+}{\Delta \Vdash \quad A_1^+ \oplus A_2^+} \qquad\qquad \frac{\Delta \Vdash \quad A_2^+}{\Delta \Vdash \quad A_1^+ \oplus A_2^+}$$

Example: negative function patterns.

$$\frac{\Delta \Vdash \quad A_1^+ \quad \Delta \Vdash \quad A_2 > \gamma}{\Delta \Vdash \quad (A_1^+ \to A_2^-) > \gamma}$$

Higher-Order Focusing

Patterns are the fulcrum of the type theory:

$$\Delta \Vdash p^+ : A^+ \qquad\qquad \Delta \Vdash p^- : A^- > \gamma$$

Example: positive disjunction patterns.

$$\frac{\Delta \Vdash p_1^+ : A_1^+}{\Delta \Vdash \mathsf{inl} \circ p_1^+ : A_1^+ \oplus A_2^+} \qquad \frac{\Delta \Vdash p_2^+ : A_2^+}{\Delta \Vdash \mathsf{inr} \circ p_2^+ : A_1^+ \oplus A_2^+}$$

Example: negative function patterns.

$$\frac{\Delta \Vdash \quad A_1^+ \quad \Delta \Vdash \quad A_2 > \gamma}{\Delta \Vdash \quad (A_1^+ \to A_2^-) > \gamma}$$

Higher-Order Focusing

Patterns are the fulcrum of the type theory:

$$\Delta \Vdash p^+ : A^+ \qquad\qquad \Delta \Vdash p^- : A^- > \gamma$$

Example: positive disjunction patterns.

$$\frac{\Delta \Vdash p_1^+ : A_1^+}{\Delta \Vdash \mathsf{inl} \circ p_1^+ : A_1^+ \oplus A_2^+} \qquad \frac{\Delta \Vdash p_2^+ : A_2^+}{\Delta \Vdash \mathsf{inr} \circ p_2^+ : A_1^+ \oplus A_2^+}$$

Example: negative function patterns.

$$\frac{\Delta \Vdash p_1^+ : A_1^+ \quad \Delta \Vdash p_2^- : A_2 > \gamma}{\Delta \Vdash \mathsf{ap}(p_1^+); p_2^- : (A_1^+ \to A_2^-) > \gamma}$$

Higher-Order Focusing

Example: positive shift pattern ends matching.

$$\frac{}{A^- \Vdash \quad \downarrow A^-}$$

Example: positive product patterns.

$$\frac{\Delta_1 \Vdash \quad A_1^+ \quad \Delta_2 \Vdash \quad A_2^+}{\Delta_1\, \Delta_2 \Vdash \quad A_1^+ \otimes A_2^+}$$

Patterns are linear: no repeated negative variables.

Higher-Order Focusing

Example: positive shift pattern ends matching.

$$\frac{}{x : A^{-} \Vdash x : \downarrow A^{-}}$$

Example: positive product patterns.

$$\frac{\Delta_1 \Vdash p_1^{+} : A_1^{+} \quad \Delta_2 \Vdash p_2^{+} : A_2^{+}}{\Delta_1\, \Delta_2 \Vdash \langle p_1^{+}, p_2^{+} \rangle : A_1^{+} \otimes A_2^{+}}$$

Patterns are linear: no repeated negative variables.

Higher-Order Focusing

Positive right focus = instantiate a value pattern:

$$\frac{\Delta \Vdash \quad A^+ \quad \Gamma \vdash \quad \Delta}{\Gamma \vdash \quad A^+}$$

Positive left inversion = analyze all patterns:

$$\frac{\Delta \Vdash \quad A^+ \quad \longrightarrow \quad \Gamma, \Delta \vdash_0 \quad \gamma}{\Gamma \vdash \quad A^+ > \gamma}$$

Premise is a meta-function mapping patterns to computations:

$$\forall\ \Delta \Vdash A^+\ \exists\ \Gamma, \Delta \vdash_0 \gamma$$

Higher-Order Focusing

Positive right focus = instantiate a value pattern:

$$\frac{\Delta \Vdash p^+ : A^+ \quad \Gamma \vdash \sigma : \Delta}{\Gamma \vdash p^+[\sigma] : A^+}$$

Positive left inversion = analyze all patterns:

$$\frac{\Delta \Vdash \quad A^+ \quad \longrightarrow \quad \Gamma, \Delta \vdash_0 \quad \gamma}{\Gamma \vdash \quad A^+ > \gamma}$$

Premise is a meta-function mapping patterns to computations:

$$\forall\ \Delta \Vdash A^+\ \exists\ \Gamma, \Delta \vdash_0 \gamma$$

Higher-Order Focusing

Positive right focus = instantiate a value pattern:

$$\frac{\Delta \Vdash p^+ : A^+ \quad \Gamma \vdash \sigma : \Delta}{\Gamma \vdash p^+[\sigma] : A^+}$$

Positive left inversion = analyze all patterns:

$$\frac{\Delta \Vdash p^+ : A^+ \quad \longrightarrow \quad \Gamma, \Delta \vdash_0 \phi^+(p) : \gamma}{\Gamma \vdash \mathsf{case}(\phi^+) : A^+ > \gamma}$$

Premise is a meta-function mapping patterns to computations:

$$\forall\ \Delta \Vdash A^+\ \exists\ \Gamma, \Delta \vdash_0 \gamma$$

Higher-Order Focusing

Example: $x{:}A^{-} \vdash \mathsf{case}(\phi^{+}) : (\mathbf{1} \oplus (\mathbf{1} \oplus \downarrow A^{-})) > \gamma$.

$$\phi^{+} : \begin{cases} \mathsf{inl} \circ \langle\rangle & \mapsto m_1 \\ \mathsf{inr} \circ \mathsf{inl} \circ \langle\rangle & \mapsto m_2 \\ \mathsf{inr} \circ \mathsf{inr} \circ x & \mapsto m_3 \end{cases}$$

One case for each possible pattern: may even be infinitary!

Higher-Order Focusing

Negative left focus = instantiate an experiment pattern:

$$\frac{\Delta \Vdash \quad A^- > \gamma_0 \quad \Gamma \vdash \qquad \Delta \quad \Gamma \vdash \qquad \gamma_0 > \gamma}{\Gamma \vdash \qquad\qquad A^- > \gamma}$$

Negative right inversion = analyze all experiments:

$$\frac{\Delta \Vdash \quad A^- > \gamma \quad \longrightarrow \quad \Gamma, \Delta \vdash \qquad : \gamma}{\Gamma \vdash \qquad A^-}$$

Negative right inversion rule involves meta-function, dually to positive left inversion.

Higher-Order Focusing

Negative left focus = instantiate an experiment pattern:

$$\frac{\Delta \Vdash p^- : A^- > \gamma_0 \quad \Gamma \vdash \sigma : \Delta \quad \Gamma \vdash k^+ : \gamma_0 > \gamma}{\Gamma \vdash p^-[\sigma]; k^+ : A^- > \gamma}$$

Negative right inversion = analyze all experiments:

$$\frac{\Delta \Vdash p^- : A^- > \gamma \quad \longrightarrow \quad \Gamma, \Delta \vdash \phi^-(p^-) : \gamma}{\Gamma \vdash \mathsf{case}(\phi^-) : A^-}$$

Negative right inversion rule involves meta-function, dually to positive left inversion.

Higher-Order Focusing

Neutral computations may have effects, structured monadically:

- Unit computation = return a value:

$$\frac{\Gamma \vdash \quad A^+}{\Gamma \vdash_0 \quad A^+}$$

- Composition of computations:

$$\frac{A^- \in \Gamma \quad \Gamma \vdash \quad A^- > \gamma}{\Gamma \vdash_0 \quad \gamma}$$

Zeilberger: distinct focused forms are observationally distinct, given enough effects.

Higher-Order Focusing

Neutral computations may have effects, structured monadically:

- Unit computation = return a value:

$$\frac{\Gamma \vdash v^+ : A^+}{\Gamma \vdash_0 \mathsf{ret}(v^+) : A^+}$$

- Composition of computations:

$$\frac{A^- \in \Gamma \quad \Gamma \vdash \qquad A^- > \gamma}{\Gamma \vdash_0 \qquad\qquad \gamma}$$

Zeilberger: distinct focused forms are observationally distinct, given enough effects.

Higher-Order Focusing

Neutral computations may have effects, structured monadically:

- Unit computation = return a value:

$$\frac{\Gamma \vdash v^+ : A^+}{\Gamma \vdash_0 \mathsf{ret}(v^+) : A^+}$$

- Composition of computations:

$$\frac{x : A^- \in \Gamma \quad \Gamma \vdash k^- : A^- > \gamma}{\Gamma \vdash_0 x \bullet k^- : \gamma}$$

Zeilberger: distinct focused forms are observationally distinct, given enough effects.

Higher-Order Focusing

Various cut principles are admissible, for example

$$\frac{\Gamma \vdash \quad A^+ \quad \Gamma \vdash \quad A^+ > \gamma}{\Gamma \vdash \quad \gamma}$$

Cut elimination yields an intrinsically safe operational semantics:

Higher-order inversion rule ensures that ϕ^+ responds to all possible values, so execution cannot "get stuck."

Higher-Order Focusing

Various cut principles are admissible, for example

$$\frac{\Gamma \vdash v^+ : A^+ \quad \Gamma \vdash k^+ : A^+ > \gamma}{\Gamma \vdash v^+ \bullet k^+ : \gamma}$$

Cut elimination yields an intrinsically safe operational semantics:

Higher-order inversion rule ensures that ϕ^+ responds to all possible values, so execution cannot "get stuck."

Higher-Order Focusing

Various cut principles are admissible, for example

$$\frac{\Gamma \vdash v^+ : A^+ \quad \Gamma \vdash k^+ : A^+ > \gamma}{\Gamma \vdash v^+ \bullet k^+ : \gamma}$$

Cut elimination yields an intrinsically safe operational semantics:

$$(p^+[\sigma] \bullet \mathsf{case}(\phi^+)) \longmapsto \phi^+(p^+)[\sigma]$$

Higher-order inversion rule ensures that ϕ^+ responds to all possible values, so execution cannot "get stuck."

Polarization

Polarize to expose operational distinctions.

$$\Gamma \vdash_{\mathsf{ML}} e : \tau \quad \rightsquigarrow \quad e^{\mathsf{ML}} : \Gamma^{\mathsf{ML}} > \tau^{\mathsf{ML}}$$
$$\Gamma \vdash_{\mathsf{H}} e : \tau \quad \rightsquigarrow \quad \Gamma^{\mathsf{H}} \vdash e^{\mathsf{H}} : \tau^{\mathsf{H}}.$$

$$\begin{aligned}
\mathsf{unit}^{\mathsf{ML}} &= \mathbf{1} & \mathsf{unit}^{\mathsf{H}} &= \top \\
(\tau_1 * \tau_2)^{\mathsf{ML}} &= \tau_1^{\mathsf{ML}} \otimes \tau_2^{\mathsf{ML}} & (\tau_1, \tau_2)^{\mathsf{H}} &= \tau_1^{\mathsf{H}} \,\&\, \tau_2^{\mathsf{H}} \\
\mathsf{void}^{\mathsf{ML}} &= \mathbf{0} & \mathsf{void}^{\mathsf{H}} &= \uparrow\downarrow \mathbf{0} \\
(\tau_1 + \tau_2)^{\mathsf{ML}} &= \tau_1^{\mathsf{ML}} \oplus \tau_2^{\mathsf{ML}} & (\tau_1 + \tau_2)^{\mathsf{H}} &= \uparrow (\downarrow \tau_1^{\mathsf{H}} \oplus \downarrow \tau_2^{\mathsf{H}}) \\
(\tau_1 \to \tau_2)^{\mathsf{ML}} &= \downarrow (\tau_1^{\mathsf{ML}} \to \uparrow \tau_2^{\mathsf{ML}}) & (\tau_1 \to \tau_2)^{\mathsf{H}} &= (\downarrow \tau_1^{\mathsf{H}}) \to \tau_2^{\mathsf{H}}
\end{aligned}$$

Applications of Focusing

Focusing has many applications in PL design and semantics!

Applications of Focusing

Focusing has many applications in PL design and semantics!

- Operationally sensitive typing: intersections and unions.

Applications of Focusing

Focusing has many applications in PL design and semantics!

- Operationally sensitive typing: intersections and unions.
- Deciding full equivalence for finite typed λ-calculus.

Applications of Focusing

Focusing has many applications in PL design and semantics!

- Operationally sensitive typing: intersections and unions.
- Deciding full equivalence for finite typed λ-calculus.
- Categorial semantics with link to Levy's call-by-push-value.

Applications of Focusing

Focusing has many applications in PL design and semantics!

- Operationally sensitive typing: intersections and unions.
- Deciding full equivalence for finite typed λ-calculus.
- Categorial semantics with link to Levy's call-by-push-value.
- Dependent types in the presence of effects.

Applications of Focusing

Focusing has many applications in PL design and semantics!

- Operationally sensitive typing: intersections and unions.
- Deciding full equivalence for finite typed λ-calculus.
- Categorial semantics with link to Levy's call-by-push-value.
- Dependent types in the presence of effects.
- Computation with binding and scope.

Applications of Focusing

Focusing has many applications in PL design and semantics!

- Operationally sensitive typing: intersections and unions.
- Deciding full equivalence for finite typed λ-calculus.
- Categorial semantics with link to Levy's call-by-push-value.
- Dependent types in the presence of effects.
- Computation with binding and scope.

(Not to mention many applications in proof theory!)

Binding and Computation

Goal: datatype mechanism with binding and computation.

- LF-style representations of syntactic objects.
- ML-style computation by structural induction.
- cf Beluga [Pientka and Dunfield], Delphin [Schuermann], FreshML [Pitts, et al.; Pottier].

Focusing provides a natural framework for integrating these!

- Binding = positive function space.
- Computation = negative function space.

Binding and Computation

The key is to integrate two forms of entailment:

- Derivability: $J_1, \ldots, J_n \vdash J$.
- Admissibility: $J_1, \ldots, J_n \models J$.

Derivability expresses binding and scope:

$$\underbrace{u_1\,\mathsf{exp}, \ldots, u_n\,\mathsf{exp}}_{\Psi} \vdash e\,\mathsf{exp}$$

Admissibility expresses computation:

$$e\,\mathsf{exp} \models \mathsf{sz}(e)\,\mathsf{nat}$$

Judgements and Evidence

Basic judgements J are inductively defined assertions.

- $e\,\mathsf{exp}$, $e : \tau$, $e \hookrightarrow e'$, ...
- Defined by a collection of rules.

Evidence for J is a derivation, $\nabla : J$.

- Consists of a composition of rules.
- Ending with judgement J.

Derivability

Derivability judgement $J_1 \vdash J_2$.

- J_2 is derivable from J_1.
- J_1 is a local axiom, or hypothesis.

Evidence for $J_1 \vdash J_2$ has the form $u.\nabla$, where

- ∇ is a derivation of J_2 involving ...
- ... the local rule, u, deriving J_1.

Schroeder-Heister: derivability may be iterated.

- $J_1 \vdash (J_2 \vdash J_3)$ equivalent to $J_1, J_2 \vdash J_3$.
- $(J_1 \vdash J_2) \vdash J$: assume a rule $J_1 \vdash J_2$ while deriving J.

Higher-Order Abstract Syntax

Example: untyped λ-terms.

$$\underbrace{u_1\,\mathsf{exp}, \ldots, u_n\,\mathsf{exp}}_{\Psi} \vdash e\,\mathsf{exp}$$

$$\frac{}{\Psi, u\,\mathsf{exp} \vdash u\,\mathsf{exp}} \qquad \frac{\Psi \vdash e_1\,\mathsf{exp} \quad \Psi \vdash e_2\,\mathsf{exp}}{\Psi \vdash \mathsf{ap}(e_1, e_2)\,\mathsf{exp}}$$

$$\frac{\Psi, u\,\mathsf{exp} \vdash e\,\mathsf{exp}}{\Psi \vdash \lambda(u.e)\,\mathsf{exp}}$$

Pronominal: choice of parameter does not matter!

- Fiore, Plotkin, Tiuri: pre-sheaves.
- Gabbay, Pitts: FM sets, equivariance.

Higher-Order Abstract Syntax

Example: untyped λ-terms.

$$\underbrace{u_1\,\mathsf{exp}, \ldots, u_n\,\mathsf{exp}}_{\Psi} \vdash e\,\mathsf{exp}$$

$$\frac{}{\Psi, u\,\mathsf{exp} \vdash u\,\mathsf{exp}} \qquad \frac{\Psi \vdash e_1\,\mathsf{exp} \quad \Psi \vdash e_2\,\mathsf{exp}}{\Psi \vdash \mathsf{ap}(e_1, e_2)\,\mathsf{exp}}$$

$$\frac{\Psi, u'\,\mathsf{exp} \vdash [u'/u]e\,\mathsf{exp}}{\Psi \vdash \lambda(u.e)\,\mathsf{exp}}$$

Pronominal: choice of parameter does not matter!

- Fiore, Plotkin, Tiuri: pre-sheaves.
- Gabbay, Pitts: FM sets, equivariance.

Admissibility

Admissibility judgement $J_1 \models J_2$.

- If J_1 is derivable, then J_2 is also derivable.
- May hold vacuously.
- Negation $\neg J$ of J: $J \models \#$.

Evidence is a meta-function on derivations.

- $\nabla : J_1 \quad \longmapsto \quad \phi(\nabla) : J_2$.
- Constructively, the transformation ϕ is computable.

Admissibility and Derivability

Expressing computations over syntax with binding mixes entailments:

$$\underbrace{u_1\,\mathsf{exp}, \ldots, u_n\,\mathsf{exp}}_{\Psi} \vdash (e\,\mathsf{exp} \models \mathsf{sz}(e)\,\mathsf{nat})$$

Admissibility and Derivability

Expressing computations over syntax with binding mixes entailments:

$$\underbrace{u_1\,\mathsf{exp}, \ldots, u_n\,\mathsf{exp}}_{\Psi} \vdash (e\,\mathsf{exp} \models \mathsf{sz}(e)\,\mathsf{nat})$$

Spelled out in words, this judgement means

- given expression variables $u_1, \ldots, u_n, \ldots$

Admissibility and Derivability

Expressing computations over syntax with binding mixes entailments:

$$\underbrace{u_1\,\mathsf{exp}, \ldots, u_n\,\mathsf{exp}}_{\Psi} \vdash (e\,\mathsf{exp} \models \mathsf{sz}(e)\,\mathsf{nat})$$

Spelled out in words, this judgement means

- given expression variables $u_1, \ldots, u_n$, ...
- if $e\,\mathsf{exp}$ is an expression, ...

Admissibility and Derivability

Expressing computations over syntax with binding mixes entailments:

$$\underbrace{u_1\,\mathsf{exp}, \ldots, u_n\,\mathsf{exp}}_{\Psi} \vdash (e\,\mathsf{exp} \models \mathsf{sz}(e)\,\mathsf{nat})$$

Spelled out in words, this judgement means

- given expression variables $u_1, \ldots, u_n$, ...
- if $e\,\mathsf{exp}$ is an expression, ...
- the size of e exists as a natural number.

Admissibility and Derivability

Evidence consists of a function S such that

$$\begin{array}{rcl} S\ (\Psi, u)\ u & \mapsto & 1 \\ S\ \Psi\ \mathsf{ap}(e_1, e_2) & \mapsto & 1 + (S\ \Psi\ e_1) + (S\ \Psi\ e_2) \\ S\ \Psi\ \lambda(u.e) & \mapsto & 1 + (S\ (\Psi, u)\ e) \end{array}$$

Defined by pattern matching against derivations!

- Abstracted over parameters Ψ.
- Parameters are extended in the recursion.

Representing Judgements and Evidence

Basic derivations are are positive values.

- Inductively generated by rules, or constructors.
- Inductively analyzed by pattern matching and recursion.

Derivability judgements are positive functions $J_1 \Rightarrow J_2$.

- A value of type J_2 with a rule constructor u of type J_1.
- Closed-ended: derivation schemas, not computations!

Admissibility judgements are negative functions $J_1 \longrightarrow J_2$.

- A (computable) transformation from J_1 to J_2.
- Open-ended: arbitrary transformation.

Contextualization

Key technique: contextual modality $\langle \Psi \rangle A^+$ [Nanevski, Pientka].

- Internalizes derivability from assumptions $R_1, \ldots, R_n$.
- Ψ is a rule context $u_1 : R_1, \ldots, u_n : R_n$.
- Each R is $D \Leftarrow A_1^+, \ldots, A_n^+$, where D is a pronominal data type.

Contextualized typing judgements:

- Right focus: $\Gamma \vdash v^+ : \langle \Psi \rangle A^+$.
- Left inversion: $\Gamma \vdash k^+ : \langle \Psi \rangle A^+ > \gamma$.

Pronominal Data Types

Pronominal data types, D, are inductively defined by context Ψ.

- Rules generate values of the type.
- Rules are extensible within a scope.

Contextualized types track the scopes of parameters.

- Enforces proper scoping of names.
- cf nominal data types, which do not.

Pronominal Data Types

Patterns:

$$\frac{D \Leftarrow A^+ \in \Psi \quad \Delta \Vdash \qquad \langle \Psi \rangle A^+}{\Delta \Vdash \qquad \langle \Psi \rangle D}$$

Values (derivable):

$$\frac{D \Leftarrow A^+ \in \Psi \quad \Gamma \vdash \qquad \langle \Psi \rangle A^+}{\Gamma \vdash \qquad \langle \Psi \rangle D}$$

Matches (derivable):

$$\frac{(\quad D \Leftarrow A^+ \in \Psi \quad \wedge \quad \Delta \Vdash \qquad A^+) \quad \longrightarrow \quad \Gamma \vdash_0 \qquad \gamma}{\Gamma \vdash \qquad\qquad\qquad\qquad\qquad \langle \Psi \rangle D > \gamma}$$

Pronominal Data Types

Patterns:

$$\frac{u{:}D{\Leftarrow}A^{+} \in \Psi \quad \Delta \Vdash p^{+} : \langle \Psi \rangle A^{+}}{\Delta \Vdash u(p^{+}) : \langle \Psi \rangle D}$$

Values (derivable):

$$\frac{D{\Leftarrow}A^{+} \in \Psi \quad \Gamma \vdash \quad \langle \Psi \rangle A^{+}}{\Gamma \vdash \quad \langle \Psi \rangle D}$$

Matches (derivable):

$$\frac{(\quad D \Leftarrow A^{+} \in \Psi \quad \wedge \quad \Delta \Vdash \quad A^{+}) \quad \longrightarrow \quad \Gamma \vdash_{0} \quad \gamma}{\Gamma \vdash \quad \langle \Psi \rangle D > \gamma}$$

Pronominal Data Types

Patterns:

$$\frac{u{:}D{\Leftarrow}A^+ \in \Psi \quad \Delta \Vdash p^+ : \langle\Psi\rangle A^+}{\Delta \Vdash u(p^+) : \langle\Psi\rangle D}$$

Values (derivable):

$$\frac{u{:}D{\Leftarrow}A^+ \in \Psi \quad \Gamma \vdash v^+ : \langle\Psi\rangle A^+}{\Gamma \vdash u(v^+) : \langle\Psi\rangle D}$$

Matches (derivable):

$$\frac{(\quad D \Leftarrow A^+ \in \Psi \quad \wedge \quad \Delta \Vdash \quad A^+) \quad \longrightarrow \quad \Gamma \vdash_0 \quad \gamma}{\Gamma \vdash \qquad\qquad\qquad\qquad \langle\Psi\rangle D > \gamma}$$

Pronominal Data Types

Patterns:

$$\frac{u{:}D{\Leftarrow}A^{+} \in \Psi \quad \Delta \Vdash p^{+} : \langle \Psi \rangle A^{+}}{\Delta \Vdash u(p^{+}) : \langle \Psi \rangle D}$$

Values (derivable):

$$\frac{u{:}D{\Leftarrow}A^{+} \in \Psi \quad \Gamma \vdash v^{+} : \langle \Psi \rangle A^{+}}{\Gamma \vdash u(v^{+}) : \langle \Psi \rangle D}$$

Matches (derivable):

$$\frac{(u{:}D \Leftarrow A^{+} \in \Psi \quad \wedge \quad \Delta \Vdash p^{+} : A^{+}) \quad \longrightarrow \quad \Gamma \vdash_0 m : \gamma}{\Gamma \vdash \mathsf{case}(\cdots \mid u(p^{+}) \mapsto m \mid \ldots) : \langle \Psi \rangle D > \gamma}$$

Positive Functions

Positive function type $R \Rightarrow A^+$:

$$\frac{\Delta \Vdash \qquad \langle \Psi, u{:}R \rangle A^+}{\Delta \Vdash \qquad \langle \Psi \rangle (R \Rightarrow A^+)}$$

Matching for positive function types:

$$\frac{\Delta \Vdash \qquad \langle \Psi, \quad R \rangle A^+ \quad \longrightarrow \quad \Gamma, \Delta \vdash_0 \qquad \gamma}{\Gamma \vdash \qquad\qquad\qquad\qquad\qquad \langle \Psi \rangle (R \Rightarrow A^+) > \gamma}$$

The parameter u is a pronoun, not a noun!

Positive Functions

Positive function type $R \Rightarrow A^+$:

$$\frac{\Delta \Vdash v^+ : \langle \Psi, u{:}R \rangle A^+}{\Delta \Vdash u.v^+ : \langle \Psi \rangle (R \Rightarrow A^+)}$$

Matching for positive function types:

$$\frac{\Delta \Vdash \quad \langle \Psi, \quad R \rangle A^+ \quad \longrightarrow \quad \Gamma, \Delta \vdash_0 \quad \gamma}{\Gamma \vdash \quad \langle \Psi \rangle (R \Rightarrow A^+) > \gamma}$$

The parameter u is a pronoun, not a noun!

Positive Functions

Positive function type $R \Rightarrow A^+$:

$$\frac{\Delta \Vdash v^+ : \langle \Psi, u{:}R \rangle A^+}{\Delta \Vdash u.v^+ : \langle \Psi \rangle (R \Rightarrow A^+)}$$

Matching for positive function types:

$$\frac{\Delta \Vdash p^+ : \langle \Psi, u{:}R \rangle A^+ \quad \longrightarrow \quad \Gamma, \Delta \vdash_0 m : \gamma}{\Gamma \vdash \mathsf{case}(u.p^+ \mapsto m) : \langle \Psi \rangle (R \Rightarrow A^+) > \gamma}$$

The parameter u is a pronoun, not a noun!

Higher-Order Syntax, Revisited

Rule context Ψ_{exp} declares constructors:

$$\begin{array}{rcl} \mathsf{ap} & : & \mathsf{exp} \Leftarrow \mathsf{exp}, \mathsf{exp} \\ \lambda & : & \mathsf{exp} \Leftarrow (\mathsf{exp} \Rightarrow \mathsf{exp}) \end{array}$$

Rule context Ψ_{var} declares expression variables:

$$u_1 : \mathsf{exp}, \ldots, u_n : \mathsf{exp}$$

Adequacy: the type $\langle \Psi_{\mathsf{exp}}\, \Psi_{\mathsf{var}} \rangle \mathsf{exp}$ internalizes the judgement

$$u_1\, \mathsf{exp}, \ldots, u_n\, \mathsf{exp} \vdash e\, \mathsf{exp}$$

Computing With Binders

Define $\mathsf{sz} = \mathsf{case}(\phi^{\text{-}})$ of negative function type

$$\langle \Psi_{\mathsf{exp}} \rangle \, \Psi_{\mathsf{var}} \Rightarrow (\mathsf{exp} \longrightarrow \mathsf{nat})$$

The meta-function $\phi^{\text{-}}$ is defined by

$$\begin{array}{rcl} \phi^{\text{-}} \ (\Psi, u{:}\mathsf{exp}) \ u & = & 1 \\ \phi^{\text{-}} \ \Psi \ (\mathsf{ap}(e_1, e_2)) & = & 1 + (\phi^{\text{-}} \ \Psi \ e_1) + (\phi^{\text{-}} \ \Psi \ e_2) \\ \phi^{\text{-}} \ \Psi \ (\lambda(u.e)) & = & 1 + (\phi^{\text{-}} \ (\Psi, u{:}\mathsf{exp}) \ e) \end{array}$$

The recursive call acts on the value e of contextualized type

$$\langle \Psi_{\mathsf{exp}} \, \Psi_{\mathsf{var}} \, u{:}\mathsf{exp} \rangle \mathsf{exp}$$

Normalization by Evaluation

Pronominal data types enforce scoping of bound variables.

- cf, nominal approaches, which do not (names abound).

Example: normalization by evaluation [ICFP09 forthcoming].

$$\mathsf{eval} : \langle \Psi_{\mathsf{nbe}} \rangle\ \forall \Psi\ \Psi \Rightarrow (\mathsf{exp} \longrightarrow (\mathsf{exp}\# \longrightarrow \mathsf{sem}) \longrightarrow \mathsf{sem})$$
$$\mathsf{reify} : \langle \Psi_{\mathsf{nbe}} \rangle\ \forall \Psi\ \Psi \Rightarrow (\mathsf{sem} \longrightarrow (\mathsf{exp}\# \longrightarrow \mathsf{neu}\#) \longrightarrow \mathsf{exp})$$

Normalization therefore has type

$$\langle \Psi_{\mathsf{nbe}} \rangle\ \forall \Psi\ \Psi \Rightarrow \mathsf{exp} \longrightarrow \mathsf{exp}.$$

Result involves only parameters from the input.

Rule Conjunction

Representational conjunction: $R \curlywedge A^{-}$.

$$\frac{\Delta \Vdash p^{-} : \langle \Psi, u{:}R \rangle A^{-} > \gamma}{\Delta \Vdash \mathsf{unpack}; u, p^{-} : \langle \Psi \rangle (R \curlywedge A^{-}) > \gamma}$$

Patterns are destructor patterns in an expanded rule context.

Some/Any

Representational connectives exhibit some/any equivalences:

- $\downarrow (R \curlywedge A^{-}) \approx R \Rightarrow \downarrow A^{-}$.
- $\uparrow (R \Rightarrow A^{+}) \approx R \curlywedge \uparrow A^{+}$.

Informally,

- A (destructor in an expanded context) is a destructor (in an expanded context).
- A (constructor in an expanded context) is a constructor (in an expanded context).

(Non-) Structurality

The rule context Ψ need not behave structurally!

- Reflexivity is assured: parameters are values of their type.
- Exchange and contraction are admissible.
- Weakening and transivity need not hold!

Rules may have side conditions.

$$r \quad : \quad J \Leftarrow J_1, \ldots, J_n, \neg K$$

Last premise demands that there are no derivations of K.

- eg, disequalities such as $l \neq l'$ for store lookup.

(Non-) Structurality

Negation $\neg K$ means $\downarrow (K \rightarrow\!\uparrow \mathbf{0})$.

- Circumscribes possible derivations of K.
- Witnesses absence of parameters of type K.

Hence weakening fails:

$$v^+ : \langle \Psi \rangle A^+ \qquad \not\supset \qquad v^+ : \langle \Psi\, \Psi' \rangle A^+$$

Example: $v^+ = r(v_1^+, \ldots, v_n^+, \mathsf{case}(\phi^-))$, where ϕ^- refutes K.

- Could be well-formed in context Ψ.
- Yet ill-formed in context $\Psi, u{:}K$.

(Non-) Structurality

Integrating binding and computation refutes structurality!

- Side conditions on rules may obstruct weakening, substitution.
- Structurality not always appropriate, eg assignable variables.

But structurality holds if side conditions govern subordinate types.

- eg, stratifies judgements into iterated form.
- eg, location equality is prior to expression transition.

When available, structural properties are generically definable.

- Generated from types, as in Haskell.

(Non-)Structurality

Failure of structurality implies that positive functions are not restricted forms of negative function!

- No substitution action $A^+ \Rightarrow B^+ \quad \hookrightarrow \quad \downarrow (A^+ \rightarrow\!\uparrow B^+)$.
- Cannot "cut down" negative functions to positive functions using modalities, polymorphism, etc.

When all rules are pure (no side conditions), then every positive function induces a negative function by substitution.

- An advantage of pure rule formalisms.
- But cannot scale to rules with impurities.

Shocking Equivalences

Representational connectives contradict computational intuitions!

- $R \Rightarrow (A_1^+ \oplus A_2^+) \approx (R \Rightarrow A_1^+) \oplus (R \Rightarrow A_2^+)$
- $(R \between A_1^-) \& (R \between A_2^-) \approx R \between (A_1^- \& A_2^-)$.

Informally,

- (A choice of values) involving a parameter is a choice of (values involving a parameter).
- A pair of (destructors in an expanded context) is a (pair of destructors) in an expanded context.

Summary

Focusing is a useful tool for programming language research!

- Integration of "eager" and "lazy" types.
- Supports operationally sensitive type systems.
- Point of contact between proof search and proof reduction paradigms.

Pronominal integration of binding and computation.

- Pattern matching over higher-order representations.
- Side conditions are "first-class citizens."
- Adequately expressive for many problems.

Ongoing Work

Implementation.

- A universe within Agda.
- deBruijn representations for parameters.
- Meta-functions are Agda functions.
- Serviceable for examples such as NBE.

Semantics (with Awodey, Lumsdaine, Birkedal).

- Categorical formulation of focusing largely in hand.
- Contextualization remains under investigation.

Comparison with other approaches [ICFP09 forthcoming].

- Well-known examples such as NBE.
- Relate to Beluga, Delphin, FreshML approach.

Ongoing Work

Positive Dependency [PLPV09]

- Admit $\Pi x : A_1^+ . A_2^-$ (negative) and $\Sigma x : A_1^+ . A_2^+$ (positive).
- Supports GADT-like computations over families of types.
- Relies on induction-recursion for proof theory.

Dependent rules for pronominal data types.

- Essential to achieve full power of LF.
- Straightforward, provided that side conditions are excluded.
- Admitting side conditions is problematic.

Thank You!

Questions?

α-Equivalence

Meta-functions must respect α-equivalence.

- $\phi^+(u.p^+) = \phi^+(u'.[u'/u]p^+)$.
- In Agda we rely on deBruijn representations.

But what is α-equivalence for patterns?

- A pattern may contain negative variables.

Need parameter renamings at shifts:

$$\frac{\pi : \Psi \cong \Psi'}{x : \langle \Psi' \rangle A^- \Vdash x_\pi : \langle \Psi \rangle \downarrow A^-}$$

$$\frac{\pi : \Psi' \cong \Psi}{\Delta \Vdash \mathsf{force}_\pi : \langle \Psi \rangle \uparrow A^+ > \langle \Psi' \rangle A^+}$$

Evaluation

```
eval : ∀ Ψ. Ψ ⇒ (exp → (exp # → sem) → sem)
eval[Ψ] x                 σ =  σ x
eval[Ψ] app(e1,e2)        σ =  appsem (eval[Ψ] e1 σ)
                                      (eval[Ψ] e2 σ)
eval[Ψ] lam(λx.e[x]) σ = slam φ where φ = ...

appsem : ∀ Ψ. Ψ ⇒ (sem → sem → sem)
appsem[Ψ] slam(φ) s2 = φ [·] s2
appsem[Ψ] neut(n) s2 = neut(napp(n , s2))
```

Evaluation

The semantic function φ is defined as follows:

φ : < Ψ > ($\forall$ (Ψ' $\in$ neu*). Ψ' $\Rightarrow$ sem $\rightarrow$ sem)

```
φ[Ψ'] s' = strengthen x from
    (eval[Ψ, x:exp , Ψ'] (weaken e[x] with Ψ') σ')
  where
    σ' : < Ψ, x:exp , Ψ' > (exp # → sem)
    σ' x          = weaken s' with x
    σ' (y ∈  Ψ) = weaken (σ y) with (x,Ψ')
```

Deciding Equivalence for the Finite Typed λ-Calculus

Finite Typed λ-Calculus

Problem: decide equivalence for finite typed λ-calculus.

- Types $\mathbf{0}$, $\mathbf{1}$, $A \times B$, $A + B$, $A \to B$.
- Main issue: sums as coproducts.

Universal condition on coproducts:

$$[M/x]N \equiv \mathsf{case}\, M \,\{\, \mathsf{inl}(y) \Rightarrow [\mathsf{inl}(y)/x]N \mid \mathsf{inr}(z) \Rightarrow [\mathsf{inr}(z)/x]N \}$$

Generalizes Shannon expansion for $\mathbf{2} = \mathbf{1} + \mathbf{1}$:

$$\begin{aligned} [M/x]N &\equiv \mathsf{if}\, M \,\mathsf{then}\, [\mathsf{tt}/x]N \,\mathsf{else}\, [\mathsf{ff}/x]N \\ &\equiv (M \wedge [\mathsf{tt}/x]N) \vee (\neg M \wedge [\mathsf{ff}/x]N) \end{aligned}$$

(basis for (O)BDD-based methods in verification)

Finite Typed λ-Calculus

The decidability is well-known, proved by two main methods:

- Term rewriting [Lindley TLCA 07]: compute canonical form using several confluent reduction systems.
- Normalization by evaluation [Altenkirch, Dybjer, Scott, Hofmann, et al]: interpret into model, extract canonical form.

Focusing provides an alternative proof [with Ahmad and Licata]:

- Polarize: Sums are positive, others are negative.
- Standardize: Analyze values of sum types as early as possible, eg as soon as variable comes into scope.

Finite Typed λ-Calculus

Theorem Two terms are equivalent in the finite typed λ-calculus iff their polarized, standardized forms are equivalent focused terms.

Equivalence of focused forms is extensional:

$$\frac{v : A^+ \quad \longrightarrow \quad \phi^+(v) \equiv \psi^+(v) : \gamma}{\mathsf{case}(\phi^+) \equiv \mathsf{case}(\psi^+) : A^+ > \gamma}$$

Theorem Extensional equality of focused terms is decidable.

- Essentially, only finite many values need be considered.
- Proof computes a generalized BDD for comparison.

Finite Typed λ-Calculus

Compared to rewriting:

- Similarity: purely proof-theoretic (structurally inductive).
- Difference: no reduction involved!

Compared to NBE:

- Similarity: polarized, standardized form is like a "model."
- Difference: purely proof-theoretic argument.