Focusing on Binding and Computation

Robert Harper
(with Daniel R. Licata and Noam Zeilberger)

Carnegie Mellon University

October 2008

Overview

Goal: datatype mechanism with binding and computation.

- LF-like representations of syntactic objects with binding and scope.
- ML-like computation by structural induction (modulo renaming).
- Dependent families of types indexed by such objects.

Applications:

- Security-typed languages based on proof-carrying API's.
- Mechanized metatheory via total f.p. (*cf.*, Agda, Delphin, Beluga).

Overview

Method: focusing, polarization, and contextualization.

- Zeilberger's focused polarized type theory (for operationally sensitive type systems).
- Nanevski and Pientka's contextual modal type theory for managing binding.

Key idea: distinguish positive from negative function space.

- Negative = computational = admissible.
- Positive = representational = derivable.

Judgements and Evidence

Judgements are forms of assertion.

- e expr, $e : \tau$, *etc.*.
- Defined by a collection of rules.

Evidence for a basic judgement J is a derivation ∇ consisting of a composition of rules.

- Abstract syntax trees, typing derivations, *etc.*.
- Write $\nabla : J$ to mean that ∇ is a derivation of J.

Derivability

The derivability judgement $J_1 \vdash J_2$ means J_2 is derivable from assumption J_1.

- Assumption is a local axiom.
- Evidence is a pattern, $a.\nabla$, consisting of evidence $\nabla : J_2$ involving the parameter $a : J_1$.
- Primitive rules are just assumed evidence for derivabilities.

In general, a rule

$$\frac{J_1 \quad \ldots \quad J_n}{J}$$

is derivable iff $J_1, \ldots, J_n \vdash J$.

Iterated Derivability

Left-iterated derivability $(J_1 \vdash J_2) \vdash J$ means that J is derivable from rule $J_1 \vdash J_2$.

- *cf.* Schroeder-Heister's definitional reflection
- Gives rise to higher-order rules (*cf.* LF representations).
- Evidence is a pattern with a parameter corresponding to the assumed rule.

Right-iterated derivability $J_1 \vdash (J_2 \vdash J_3)$ means $J_1, J_2 \vdash J_3$, with multiple assumptions.

Iterated Derivability

Higher-order rules arise naturally:

$$\frac{A\,\mathsf{true} \vdash B\,\mathsf{true}}{A \supset B\,\mathsf{true}}$$

Expressed as a derivability,

$$(A\,\mathsf{true} \vdash B\,\mathsf{true}) \vdash A \supset B\,\mathsf{true}$$

Derivable rules:

$$(A\,\mathsf{true} \vdash B\,\mathsf{true}) \vdash (A \wedge C\,\mathsf{true} \vdash B \wedge C\,\mathsf{true})$$

Admissibility

The admissibility judgement $J_1 \models J_2$ means that evidence for J_1 may be transformed into evidence for J_2.

- Evidence is any (computable) function sending any $\nabla_1 : J_1$ to some $\nabla_2 : J_2$.
- Typically defined by pattern matching against derivations $\nabla_1 : J_1$ to obtain $\nabla_2 : J_2$ in each case.

A rule

$$\frac{J_1 \quad \dots \quad J_n}{J}$$

is admissible iff $J_1, \dots, J_n \models J$.

Admissibility

Admissibility, being implication, is structural:

- Reflexivity: $J \models J$.
- Transitivity: if $J_1 \models J_2$ and $J_2 \models J_3$, then $J_1 \models J_3$.
- Weakening: if $J_1 \models J$, then $J_1, J_2 \models J$.
- Contraction: if $J_1, J_1 \models J$, then $J_1 \models J$.
- Exchange: if $J_1, J_2 \models J$, then $J_2, J_1 \models J$.

These could all be phrased as iterated admissibilities, *e.g.*,

$$(J_1 \models J) \models (J_1, J_2 \models J).$$

Admissibility

Admissibilities $J_1 \models J_2$ are not stable under rule extension!

- If $J_1 \models J_2$, then $J \models (J_1 \models J_2)$, but not $J \vdash (J_1 \models J_2)$.
- Why? Admissibility considers all derivations of antecedent.

Adding new rules disrupts evidence for admissibility.

- $(\mathbf{IL} \vdash \exists x.\phi\,\text{true}) \models (\mathbf{IL} \vdash \phi(t)\,\text{true})$ for some term t.
- But this fails for $\mathbf{CL} = \mathbf{IL} + \mathbf{LEM}$.

Admissibilities circumscribe the evidence for a judgement.

Admissibility

If all primitive rules are pure, then derivability is structural.

- Reflexivity: $J \vdash J$.
- Transitivity: $(J_1 \vdash J_2, J_2 \vdash J_3) \models (J_1 \vdash J_3)$.
- Weakening: $(J_1 \vdash J) \models (J_1, J_2 \vdash J)$.
- Contraction: $(J_1, J_1 \vdash J) \models (J_1 \vdash J)$.
- Exchange: $(J_1, J_2 \vdash J) \models (J_2, J_1 \vdash J)$.

Pure rules are those without side conditions, *i.e.*, without constraints on applicability.

Admissibility

Evidence for weakening transforms derivations rule-by-rule.

$$\frac{\Gamma \vdash J_1 \quad \ldots \quad \Gamma \vdash J_n}{\Gamma \vdash J}$$

That is, we pattern match on the last rule of $\nabla : \Gamma \vdash J$, and recursively transform premises and apply the same rule.

The validity of this argument depends on purity! Rule must continue to apply after transformation of premises.

Admissibility

Evidence for weakening transforms derivations rule-by-rule.

$$\frac{\Gamma\,\Gamma' \vdash J_1 \quad \ldots \quad \Gamma\,\Gamma' \vdash J_n}{\Gamma\,\Gamma' \vdash J}$$

That is, we pattern match on the last rule of $\nabla : \Gamma \vdash J$, and recursively transform premises and apply the same rule.

The validity of this argument depends on purity! Rule must continue to apply after transformation of premises.

Admissibility

Side conditions on rules may be seen as admissibility premises.

- $\neg J$ is just $J \models \#$.
- Need not be negations, but this is a common case.

Side conditions may disrupt structural properties, *e.g.*,

$$\frac{\Gamma \vdash J_1 \quad \ldots \quad \Gamma \vdash J_n \quad \Gamma \vdash \neg J}{\Gamma \vdash J}$$

Admissibility

Side conditions on rules may be seen as admissibility premises.

- $\neg J$ is just $J \models \#$.
- Need not be negations, but this is a common case.

Side conditions may disrupt structural properties, *e.g.*,

$$\frac{\Gamma\,\Gamma' \vdash J_1 \quad \ldots \quad \Gamma\,\Gamma' \vdash J_n \quad \Gamma\,\Gamma' \not\vdash \neg J}{\Gamma\,\Gamma' \vdash J}$$

Derivability and Admissibility

Two notions of entailment:

1. Derivability: introduced by patterns, eliminated by pattern matching.
2. Admissibility: introduced by any computable transformation and eliminated by application.

Intermixing these leads to a general theory of rules that accounts for side conditions, and allows us to express meta-theoretic properties such as admissibility and derivability of rules.

Polarized Types

Two views of the meaning of a logical connective:

- Verificationist: defined by introduction; elimination inverts introduction.
- Pragmatist: defined by elimination; introduction inverts elimination.

Operationally, these determine different connectives:

- Positive, or eager: values are compositions of patterns; elimination by pattern matching.
- Negative, or lazy: experiments are compositions of patterns; introduction by pattern matching.

Polarized Types

Positive type: natural numbers.

- Introduction: z, s(z), s(s(z)), ...
- Elimination:

$$\phi \text{ s.t. } \begin{cases} \mathsf{z} & \mapsto e_0 \\ \mathsf{s(z)} & \mapsto e_1 \\ \mathsf{s(s(z))} & \mapsto e_2 \\ \dots & \end{cases}$$

Crucially, elimination must cover all values!

Polarized Types

Negative type: infinite streams.

- Elimination: hd, tl.
- Introduction:

$$\sigma \text{ s.t. } \begin{cases} \mathsf{hd} & \mapsto e_0 \\ \mathsf{tl};\mathsf{hd} & \mapsto e_1 \\ \mathsf{tl};\mathsf{tl};\mathsf{hd} & \mapsto e_2 \\ \ldots \end{cases}$$

Crucially, introduction must cover all experiments!

Polarized Types

Computational (ML, Coq) functions are negative:

- Introduced by defining response to an argument, not by internal structure.
- Eliminated by application to an argument value.

Computational functions are open-ended:

- Any mapping from domain to range is acceptable.
- Pragmatically, allows us to import functions from other systems.

Polarized Types

Representational (LF) functions are positive:

- Introduced by compositions of constructors, starting with variables.
- Eliminated by pattern matching, not application.

Representational functions are closed-ended:

- Cannot enrich with operations that analyze form of input.
- Essentially a value with indeterminates.

Functions and Entailment

Positive (representational) functions witness derivability.

- Parameters are “fresh” axioms/assumptions.
- Body is a derivation schema with distinguished parameters.

Negative (computational) functions witness admissibility.

- Analyzes all possible derivations of antecedent.
- Computes a derivation for each possible argument.

Types for Binding and Computation

Polarization (Girard)

- Distinguish positive (verificationist/inductive/eager) from negative (pragmatist/coinductive/lazy) connectives.
- Investigated by Zeilberger in connection with operationally sensitive type systems (intersections and unions).

Types for Binding and Computation

Focusing (Andreoli, Girard)

- Patterns mediate between focus and inversion.
- Positive: (right) focus = choose a value, (left) invert = pattern match.
- Negative: (left) focus = choose an experiment, (right) invert = respond to experiments.

Types for Binding and Computation

Contextual Modality (Nanevski and Pientka)

- Types for managing binding and scope (*cf.*, Fiore, Tiuri, Plotkin pre-sheaf approach).
- Definitional variation for scoped rules (datatype definitions).
- Pronominal representation of binding and scope.

Pronominal representation avoids machinery of names.

- Parameters are pronouns, not nouns (names are not objects, but pointers to binding sites).
- Crucial for dependency on objects with binding (no effects).

Focusing Framework

Positive (right) focus: choose a value of positive type.

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

A value is given by a pattern under a substitution.

- Variables range only over negative types.
- Variables must be used linearly.

Focusing Framework

Positive (left) inversion: respond to all possible choices.

$$\frac{\Delta \Vdash p :: C^+ \longrightarrow \Gamma\, \Delta \vdash \phi^+(p) :: \gamma}{\Gamma \vdash \mathsf{val}(\phi^+) :: C^+ > \gamma}$$

An inversion is defined for all patterns of its domain type.

- $\phi^+ : \{\, p_0(\vec{x_0}) \mapsto e_0(\vec{x_0}) \mid p_1(\vec{x_1}) \mapsto e_1(\vec{x_1}) \mid \ldots \}$.
- Open-endedness: ϕ^+ is an arbitrary mapping!

Positive Patterns

Shifted (negative) type:

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

Positive product types:

$$\frac{}{\emptyset \Vdash \langle\rangle :: 1} \qquad \frac{\Delta_1 \Vdash p_1 :: A_1^+ \quad \Delta_2 \Vdash p_2 :: A_2^+}{\Delta_1\, \Delta_2 \Vdash \langle p_1, p_2 \rangle :: A_1^+ \times A_2^+}$$

Positive Patterns

Positive sum types:

$$\frac{\Delta \Vdash p^+ :: A_1^+}{\Delta \Vdash \mathsf{inl}(p^+) :: A_1^+ \oplus A_2^+} \qquad \frac{\Delta \Vdash p^+ :: A_2^+}{\Delta \Vdash \mathsf{inr}(p^+) :: A_1^+ \oplus A_2^+}$$

Focusing Framework

Negative (left) focus: choose an experiment.

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

Negative (right) inversion: respond to all choices.

$$\frac{\Delta \Vdash q :: C^{-} > \gamma \longrightarrow \Gamma\, \Delta \vdash \phi^{-}(q) : \gamma}{\Gamma \vdash \mathsf{val}(\phi^{-}) : C^{-}}$$

Negative Patterns

Shifted (positive) types:

$$\frac{}{\vdash \varepsilon :: \uparrow A^{+} > A^{+}}$$

Computational functions:

$$\frac{\Delta_1 \Vdash p :: A_1^{+} \quad \Delta_2 \Vdash q :: A_2^{-} > \gamma}{\Delta_1 \, \Delta_2 \Vdash p; q :: A_1^{+} \to A_2^{-} > \gamma}$$

Negative Patterns

Negative product types:

$$\frac{\Delta \Vdash q :: A_1^-}{\Delta \Vdash \mathsf{fst}; q :: A_1^- \& A_2^- > \gamma} \qquad \frac{\Delta \Vdash q :: A_2^-}{\Delta \Vdash \mathsf{snd}; q :: A_1^- \& A_2^- > \gamma}$$

Focusing Framework

An expression represents an outcome of a computation, either a positive value or an experiment on a negative variable.

$$\frac{\Gamma \vdash v^{+} : C^{+}}{\Gamma \vdash v^{+} :: C^{+}} \qquad \frac{\Gamma \vdash x : C^{-} \quad \Gamma \vdash k^{-} :: C^{-} > \gamma}{\Gamma \vdash x \bullet k^{-} : \gamma}$$

Focusing Framework

Cut principles start computations:

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

$$\frac{\Gamma \vdash v^{-} : C^{-} \quad \Gamma \vdash k^{-} : C^{-} > \gamma}{\Gamma \vdash v^{-} \bullet k^{-} : \gamma}$$

Operational semantics (cut reduction) is generic!

$$(p[\sigma]) \bullet \mathsf{val}(\phi) \hookrightarrow (\phi(p))[\sigma]$$

$$(v^{+} \bullet (k_1^{+}; k_2^{+}) \hookrightarrow (v^{+} \bullet k_1^{+}); k_2^{+}$$

Representational Functions

Representational function type, $R \Rightarrow A^+$, is positive.

- Represent derivabilities and binders.
- Patterns are patterns of type A^+ with a parameter of type R.
- Domain is limited to a class of rules.
- Occurrences of X in R are not negative!

Rules declare constructors of an abstract type (*cf.* ML datatypes).

- $R ::= X \Leftarrow A_1^+ \Leftarrow \cdots \Leftarrow A_n^+$.
- Side conditions: $A_i =\downarrow (B_i^+ \rightarrow C_i^-)$.
- Derivabilities: $A_i = R_i \Rightarrow C_i^+$.

Representational Functions

Positive patterns: $\Delta; \Psi \Vdash p :: C^+$.

- Ψ is a rule context $u_1 : R_1, \ldots, u_n : R_n$.
- Context Ψ is not necessarily structural!

Representational function: $R \Rightarrow A^+$.

$$\frac{\Delta; \Psi, u : R \Vdash p :: A^+}{\Delta; \Psi \Vdash \lambda u.p :: R \Rightarrow A^+}$$

Defined atoms:

$$\frac{\Psi \vdash u : X \Leftarrow A_1^+ \Leftarrow \cdots \Leftarrow A_n^+ \qquad \Delta; \Psi \Vdash p_1 :: A_1^+ \quad \ldots \quad \Delta; \Psi \Vdash p_n :: A_n^+}{\Delta; \Psi \Vdash u\, p_1\, \ldots\, p_n :: X}$$

Representational Conjunction

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

$$\frac{\Delta; \Psi, u : R \Vdash q :: A^{-}}{\Delta; \Psi \Vdash \text{unpack}; u.q :: R \curlywedge A^{-}}$$

Informally, an element consists of a destructor pattern 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).

Shocking Equivalences

Representational connectives contradict computational intuitions!

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

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.

Structural Properties

Structural properties for the contextual modality are not assured!

- May not validate weakening/proliferation = adding a new rule.
- May not validate transitivity/substitution = deriving a rule.
- Always validates exchange, contraction.

Impurities disrupt structural properties!

- No impurities: substitution is definable (*e.g.*, LF).
- With impurities: may or may not be definable.

Key: iterated inductive definition.

Example

A simple expression language:

$$e \quad ::= \quad \mathsf{num}[k] \mid e_1 \odot_f e_2 \mid \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2$$

Represented by context Ψ_{exp}:

$$\begin{aligned}
\mathsf{zero} &: \mathsf{nat} \\
\mathsf{succ} &: \mathsf{nat} \Leftarrow \mathsf{nat} \\
\mathsf{num} &: \mathsf{nat} \Leftarrow \mathsf{exp} \\
\mathsf{binop} &: \mathsf{exp} \Leftarrow (\mathsf{nat} \otimes \mathsf{nat} \to \mathsf{nat}) \Leftarrow \mathsf{exp} \\
\mathsf{let} &: \mathsf{exp} \Leftarrow \mathsf{exp} \Leftarrow (\mathsf{exp} \Leftarrow \mathsf{exp})
\end{aligned}$$

Example

We wish to define an evaluator for expressions:

$$\mathsf{fix}(E.ev) : \langle \Psi_{\mathsf{exp}} \rangle (\mathsf{exp} \rightarrow \mathsf{nat})$$

It suffices to show

$$\Delta \Vdash e : \langle \Psi_{\mathsf{exp}} \rangle \mathsf{exp}$$

$$\longrightarrow$$

$$E : \langle \Psi_{\mathsf{exp}} \rangle (\mathsf{exp} \rightarrow \mathsf{nat}); \Delta \vdash \mathsf{ev}(e) : \langle \Psi_{\mathsf{nat}} \rangle \mathsf{nat}$$

Example

This can be achieved by the following mapping:

$$\begin{aligned} \mathsf{num}\, n &\longmapsto n \\ \mathsf{binop}\, e_1\, f\, e_2 &\longmapsto f\,(E\, e_1)\,(E\, e_2) \\ \mathsf{let}\, e_1\, (\lambda u.e_2) &\longmapsto E(\mathsf{subst}\, \lambda u.e_2\, e_1) \end{aligned}$$

The computational function subst witnesses admissibility of transitivity for Ψ_{exp}.

- Exists because rules form an iterated inductive definition.
- Defined by pattern matching on $\lambda u.e_2$.

Future Work

Implementation:

- Currently, represented within Agda.
- Ongoing, design of a concrete language for meta-functions.

Enriched rule formalism:

- Extension to full LF, but without impurities.
- Can we admit impurities (*i.e.*, LF with ML)?

Positive dependent types.

- Admit $\Pi x : A_1^+ . A_2^-$ (negative) and $\Sigma x : A_1^+ . A_2^+$ (positive).
- Avoid testing equivalence of negative values.
- Simultaneous induction-recursion.

Thank You!

Questions?