

Normalization by Evaluation for the Calculus of Constructions

Andreas Abel

Department of Computer Science
Ludwig-Maximilians-University Munich

TYPES Workshop
Aussois, France
13 May 2009

Building η into Definitional Equality

- Coq's definitional equality is β (+ δ + ι).
- The stronger definitional equality, the fewer the user has to revert to equality proofs.
- Why not η ? ($f = \lambda x. f x$ if x new)
- Validates, for instance, $f = \text{comp } f \text{ id}$.
- This work:
 - 1 Efficient algorithm for deciding definitional equality.
 - 2 Correctness proof by model for CoC.

Eta laws

- Function type.

$$\frac{\vdash t : T \rightarrow U}{\vdash t = \lambda x : T. tx : T \rightarrow U} \quad x \notin \text{FV}(t)$$

- Unit type.

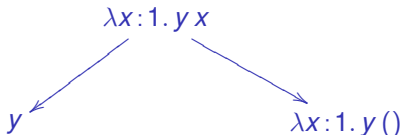
$$\frac{\vdash t : 1}{\vdash t = () : 1}$$

- Identity type: proof irrelevance.

$$\frac{\vdash p, q : \text{Id}_T t t'}{\vdash p = q : \text{Id}_T t t'} \quad \frac{\vdash p : \text{Id}_T t t}{\vdash p = \text{refl} : \text{Id}_T t t}$$

Eta reduction

- η reduction $\lambda x. t x \longrightarrow_{\eta} t$ is a program optimization.
- Not suited for deciding equality, problems abound:
 - 1 Subject reduction fails in Curry-style System F without subtyping.
 - 2 Confluence fails for surjective pairing $(fst\ t, snd\ t) \longrightarrow_{\eta} t$ (untyped).
 - 3 The unit type has no η reduction.
 - 4 Mixing reduction and expansion breaks local confluence.



- Applying rewriting to η reduction ...

Rewriting η reduction

η -reduction



Eta expansion

- Eta *expansion* works for all types with at most one introduction.

$$t \xrightarrow{U \rightarrow T} \lambda x : U. t x$$

$$t \xrightarrow{1} ()$$

$$t \xrightarrow{\text{Id}_U \ u \ u} \text{refl}$$

- Also: Σ , singleton, record, empty type.

Eta expansion and strong normalization

- Do not η -expand introductions!

$$\lambda x:U. t \xrightarrow{U \rightarrow T} \lambda y:U. (\lambda x:U. t) y \xrightarrow{\beta} \lambda y:U. t[y/x] =_{\alpha} \lambda x:U. t$$

- Do not η -expand things in elimination!

$$t^{U \rightarrow T} x \xrightarrow{T} (\lambda x:U. t x) x \xrightarrow{\beta} t x$$

- Do not η -expand at non- β -normal types! (di Cosmo)

$$x \xrightarrow{(\lambda_-(X \rightarrow X). X)x \rightarrow X} \lambda y: (\lambda_-(X \rightarrow X). X)x. x y$$

Slicing SN for η expansion

 $SN_{\beta\eta}$ 

The Truth

Eta needs a strategy!

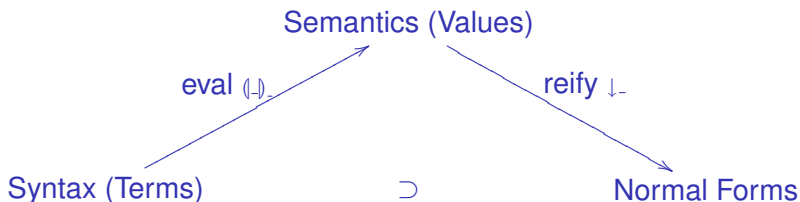
I suggest: normalization by evaluation.

The Truth

Eta needs a strategy!

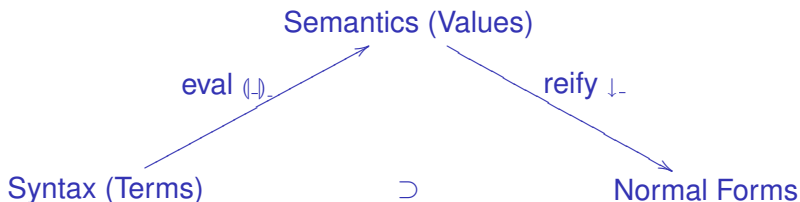
I suggest: normalization by evaluation.

What is Normalization By Evaluation?



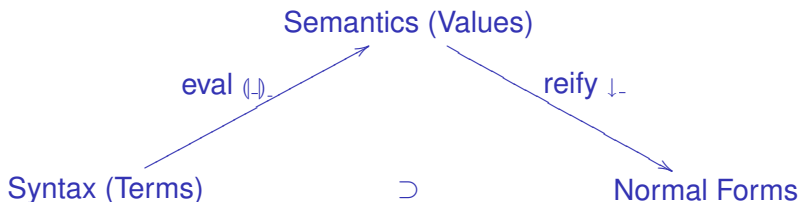
- You have: an interpreter (\Vdash) .
- You buy: a reifyer (\downarrow) .
- You get for free: a *full normalizer*!
- Famous instance: Leroy/Gregoire's compiled reduction.

What is Normalization By Evaluation?



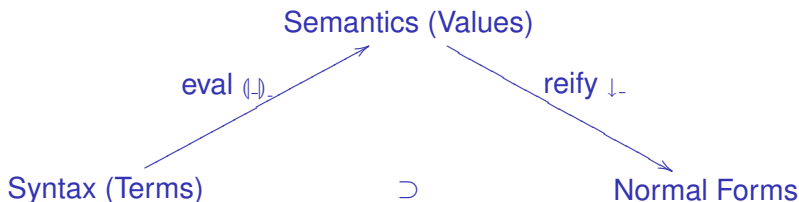
- You have: an interpreter (\Vdash) .
- You buy: a reifyer (\downarrow) .
- You get for free: a *full normalizer*!
- Famous instance: Leroy/Gregoire's compiled reduction.

What is Normalization By Evaluation?



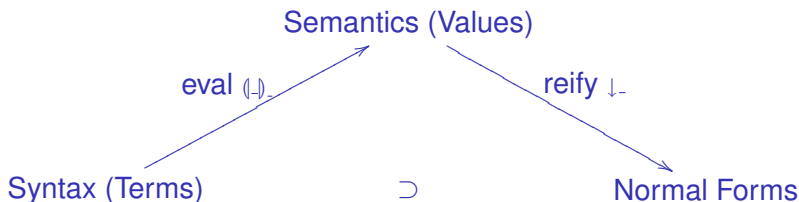
- You have: an interpreter $(|-)$.
- You buy: a reifyer $(\downarrow -)$.
- You get for free: a *full normalizer*!
- Famous instance: Leroy/Gregoire's compiled reduction.

What is Normalization By Evaluation?



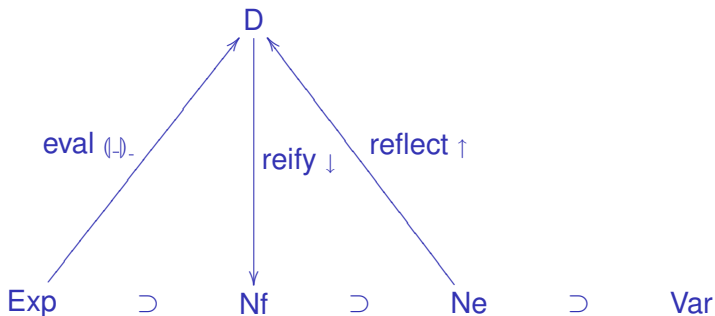
- You have: an interpreter $(|_)_$.
- You buy: a reifyer $(\downarrow_)$.
- You get for free: a *full normalizer*!
- Famous instance: Leroy/Gregoire's compiled reduction.

What is Normalization By Evaluation?



- You have: an interpreter $(|_)_{\cdot}$.
- You buy: a reifyer $(\downarrow _)$.
- You get for free: a *full normalizer*!
- Famous instance: Leroy/Gregoire's compiled reduction.

Reflecting Variables into the Semantics



Interpretation in Scott Domain

- Set of values $D \cong \text{Ne} + [D \rightarrow D]$.
- Application operation $_ \cdot _ : D \times D \rightarrow D$.
- Interpretation $\langle t \rangle_\eta \in D$ for term t and environment η :

$$\begin{aligned} \langle x \rangle_\eta &= \eta(x) \\ \langle r s \rangle_\eta &= \langle r \rangle_\eta \cdot \langle s \rangle_\eta \\ \langle \lambda x t \rangle_\eta \cdot d &= \langle t \rangle_{\eta[x \mapsto d]} \end{aligned}$$

Simply Typed Reification

- Reification $D \rightarrow Nf$:

$$\begin{aligned} \downarrow^{S \rightarrow T} f &= \lambda x : S. \downarrow^T (f \cdot (\uparrow^S x)) && x \text{ fresh} \\ \downarrow^* u &= u \end{aligned}$$

- Reflection $Ne \rightarrow D$:

$$\begin{aligned} (\uparrow^{S \rightarrow T} u) \cdot d &= \uparrow^T (u (\downarrow^S d)) \\ \uparrow^* u &= u \end{aligned}$$

- Freshness: Term families (Berger, Schwichtenberg et. al.), Lifiable de Bruijn terms (Aehlig et. al.), Nominal sets (Pitts), Kripke semantics, Contextual reification (Altenkirch et. al., Abel et. al.)

Values in the CoC

- $\text{Sort} \ni s ::= * \mid \square$
- $D \cong \text{Ne} + \text{Sort} + \text{lam}^{\text{Sort}} D [D \rightarrow D] + \text{fun}^{\text{Sort}} D [D \rightarrow D]$.
- Objects (programs)

d, e, f	$::=$	$x \vec{v}$	neutral object
		$\mid \text{lam}^{\square} K f$	type abstraction
		$\mid \text{lam}^* A f$	object abstraction

- Types

A, B	$::=$	$X \vec{v}$	neutral type
		$\mid \text{fun}^{\square} K F$	universal quantification
		$\mid \text{fun}^* A F$	dependent function type

Classification of Values (ctd.)

- Type constructors

F, G	$::=$	A	type
		$\text{lam}^{\square} K F$	type abstraction
		$\text{lam}^* A F$	object abstraction

- Kinds (“types of type constructors”)

K, L	$::=$	$*$	
		$\text{fun}^{\square} K L$	function kind
		$\text{fun}^* A L$	indexed kind

Reification and Reflection

- Reification $\downarrow^- _ \in [D \rightarrow [D \rightarrow \text{Nf}]]$.

$$\begin{aligned} \downarrow^{\text{fun}^s A F} f &= \lambda x : (\downarrow^s A). \downarrow^{F(\uparrow^A x)} (f \cdot (\uparrow^A x)) \\ \downarrow^{s'} \text{fun}^s A F &= \Pi x : (\downarrow^s A). \downarrow^{s'} (F(\uparrow^A x)) \\ \downarrow^- v &= v \end{aligned}$$

- Reflection $\uparrow^- _ \in [D \rightarrow [\text{Ne} \rightarrow D]]$.

$$\begin{aligned} \uparrow^{\text{fun}^s A F} u &= \text{lam}^s (\uparrow^s A) (d \mapsto \uparrow^{F(d)} (u(\downarrow^A d))) \\ \uparrow^- u &= u \end{aligned}$$

Normalization algorithm

- $\text{nbe}^T t = \downarrow^{(T)} (t)$
- Soundness: If $\vdash t : T$ then $\vdash t = \text{nbe}^T t : T$.
- Completeness: If $\vdash t = t' : T$ then $\text{nbe}^T t =_{\alpha} \text{nbe}^T t'$.
- Show completeness using a PER model:
 - 1 If $\vdash t = t' : T$ then $((t), (t')) \in \llbracket T \rrbracket$.
 - 2 If $\vdash T : *$ then $(T) \Vdash \llbracket T \rrbracket$.
 - 3 If $(d, d') \in \mathcal{A}$ and $A \Vdash \mathcal{A}$ then $\downarrow^A d =_{\alpha} \downarrow^A d'$.

Normalization for Impredicative Systems

- 1 Lay out a lattice of type candidates \mathcal{A} .
- 2 Specify their properties (e.g., CR1-3).
- 3 Replay type constructions (Π , \forall) on candidates.
- 4 Prove they preserve the properties.
- 5 Define a partial interpretation $\llbracket T \rrbracket$ of types.
- 6 Show the fundamental theorem:
 - $\vdash T : *$ implies $\llbracket T \rrbracket$ is a well-defined candidate.
 - $\vdash T = T' : *$ implies $\llbracket T \rrbracket = \llbracket T' \rrbracket$.
 - $\vdash t : T$ implies $(t) \in \llbracket T \rrbracket$.

Interpretation of Types

- A type T is interpreted by a pair (A, \mathcal{A}) .
 - ① $A \in D$
 - ② $\mathcal{A} \subseteq D \times D$ is a partial equivalence
 - ③ $A \Vdash \mathcal{A}$, meaning $\uparrow^A \in \text{Ne} \rightarrow \mathcal{A}$ and $\downarrow^A \in \mathcal{A} \rightarrow \text{Nf}$
 - ① $(\uparrow^A u, \uparrow^A u) \in \mathcal{A}$ for all $u \in \text{Ne}$
 - ② if $(d, d') \in \mathcal{A}$ then $\downarrow^A d =_\alpha \downarrow^A d' \in \text{Nf}$.
- Let $A, A' \Vdash^* \mathcal{A}$ imply $\uparrow^A = \uparrow^{A'} \in \text{Ne} \rightarrow \mathcal{A}$ and $\downarrow^A = \downarrow^{A'} \in \mathcal{A} \rightarrow \text{Nf}$
- Equality of types $(A, \mathcal{A}) = (A', \mathcal{A}')$ holds if
 - ① $\mathcal{A} = \mathcal{A}'$
 - ② $\downarrow^* A = \downarrow^* A' \in \text{Nf}$
 - ③ $A, A' \Vdash^* \mathcal{A}$

Interpretation of Type Constructors

- Kinds have a *shape* $k ::= * \mid k \rightarrow k' \mid \diamond \rightarrow k$.

$$\begin{aligned} \langle * \rangle &= \text{Per}(\mathbf{D}) \\ \langle k \rightarrow k' \rangle &= \mathbf{D} \times \langle k \rangle \rightarrow \langle k' \rangle \\ \langle \diamond \rightarrow k \rangle &= \mathbf{D} \rightarrow \langle k \rangle \end{aligned}$$

- A type constructor T of shape k is interpreted by a pair (F, \mathcal{F})

- 1 $F \in \mathbf{D}$

- 2 $\mathcal{F} \in \langle k \rangle$

- 3 $F, F \Vdash^k \mathcal{F}$

$$F, F' \Vdash^{k_1 \rightarrow k_2} \mathcal{F} \quad \text{iff} \quad F \cdot G, F' \cdot G \Vdash^{k_2} \mathcal{F}(G, G) \quad \forall (G, G) \in \text{dom}(\mathcal{F})$$

$$F, F' \Vdash^{\diamond \rightarrow k} \mathcal{F} \quad \text{iff} \quad F \cdot d, F' \cdot d \Vdash^k \mathcal{F}(d) \quad \forall d \in \text{dom}(\mathcal{F})$$

- Two constructors (F, \mathcal{F}) and (F', \mathcal{F}') of kind K are equal if

- 1 $\mathcal{F} = \mathcal{F}'$

- 2 $\downarrow^K F = \downarrow^K F'$

- 3 $F, F' \Vdash^K \mathcal{F}$

Interpretation of Kinds

- A kind with shape k is interpreted by (K, \mathcal{K})
 - 1 $K \in \mathbf{D}$
 - 2 $\mathcal{K} \subseteq \mathbf{D} \times \mathbf{D} \times \langle k \rangle$
 - 3 $K \Vdash^{\square} \mathcal{K}$ meaning
 - $(\uparrow^K u, \uparrow^K u, \perp^k) \in \mathcal{K}$ for all $u \in \mathbf{Ne}$
 - $(F, F', \mathcal{F}) \in \mathcal{K}$ implies $F = F' \Vdash^K \mathcal{F}$.
- Two kinds (K, \mathcal{K}) and (K', \mathcal{K}') are equal if ...

Semantic Constructions

- Dependent function space $\Pi \mathcal{A} \mathcal{F}$.
- Universal quantification $\forall \mathcal{K} \mathcal{F}$.

$$\forall \mathcal{K} \mathcal{F} = \{(d, d') \mid (d \cdot G, d' \cdot G') \in \mathcal{F}(G, G') \text{ for all } (G, G', \mathcal{G}) \in \mathcal{K}\}$$

$$\frac{K = K' \Vdash^{\square} \mathcal{K} \quad F(G) = F'(G') \Vdash^* \mathcal{F}(G, \mathcal{G}) \text{ for all } (G, G', \mathcal{G}) \in \mathcal{K}}{\text{fun}^{\square} K F = \text{fun}^{\square} K' F' \Vdash^* \forall \mathcal{K} \mathcal{F}}$$

- Indexed kinds $\Pi^{\text{fun}^* A L} \mathcal{A} \mathcal{L}$.
- Function kinds $\Pi^{\text{fun}^{\square} K L} \mathcal{K} \mathcal{L}$.

Conclusions

- Principled $\beta\eta$ -normalization for CoC.
- Can be extended to CIC.
- And to proof irrelevance (Abel, Coquand, Miguel, TLCA'09).
- A (small) add-on to Gregoire/Leroy's compiled reduction.
- Add bits of extensionality to Coq.
- Related work: OTT (Altenkirch, McBride, Swierstra, PLPV'07).
- Current work: η -expansion for identity type.