Normalization by Evaluation for the Calculus of Constructions

Andreas Abel

INRIA, Team πr^2
PPS Lab, Paris

PPS Seminar
Paris, France
15 October 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, $f = \mathsf{comp}\, f\, \mathsf{id}$.
- This work:
  1. Efficient algorithm for deciding definitional equality.
  2. Correctness proof by model for CoC.

Eta laws

- Function type.

$$\frac{\Gamma \vdash t : U \to T}{\Gamma \vdash t = \lambda x{:}U.t\,x : U \to T}\ x \notin \mathrm{FV}(t)$$

- Unit type.

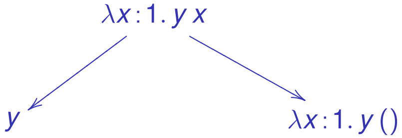
$$\frac{\Gamma \vdash t : 1}{\Gamma \vdash t = () : 1}$$

- Identity type: proof irrelevance.

$$\frac{\Gamma \vdash p, q : \mathrm{Id}_T\,t\,t'}{\Gamma \vdash p = q : \mathrm{Id}_T\,t\,t'} \qquad \frac{\Gamma \vdash p : \mathrm{Id}_T\,t\,t}{\Gamma \vdash p = \mathrm{refl} : \mathrm{Id}_T\,t\,t}$$

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 $(\mathsf{fst}\, t, \mathsf{snd}\, t) \longrightarrow_\eta t$ (untyped).
  3. The unit type has no η reduction.
  4. Mixing reduction and expansion breaks local confluence.



Eta expansion

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

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

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

$$t \xrightarrow{\mathsf{Id}_U\, u\, u} \mathsf{refl}$$

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

Eta expansion and strong normalization

- Do not η-expand introductions!

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

- Do not η-expand things in elimination!

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

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

$$x \xrightarrow{(\lambda\_:(X\to X).\,X)x\to X} \lambda y : (\lambda\_:(X \to X).\,X)x.\,x\,y$$

The Truth

Eta needs a strategy!

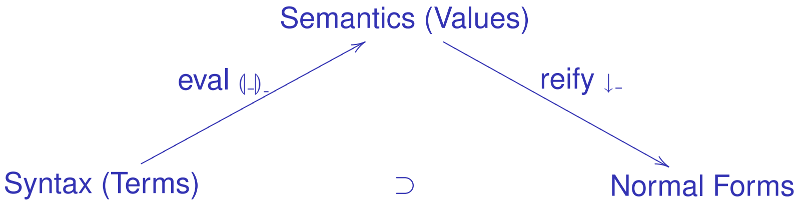
I suggest: normalization by evaluation.

The Truth

Eta needs a strategy!

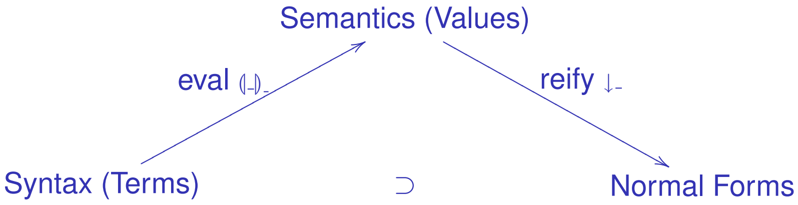
I suggest: normalization by evaluation.

What is Normalization By Evaluation?



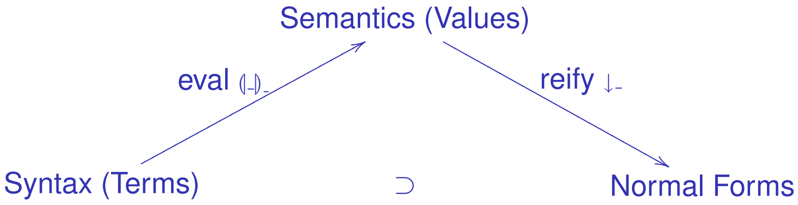
- You have: an interpreter ($(\!|\_|\!)_{\_}$).
- You buy: a reifyer ($\downarrow_{\_}$).
- You get for free: a *full normalizer*!
- Famous instance: Leroy/Gregoire's compiled reduction.

What is Normalization By Evaluation?



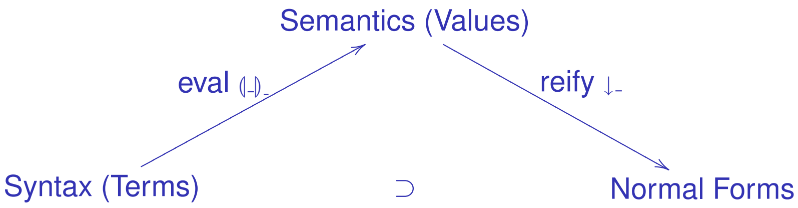
- You have: an interpreter $(\llparenthesis \_ \rrparenthesis \_)$.
- You buy: a reifyer $(\downarrow \_)$.
- You get for free: a *full normalizer*!
- Famous instance: Leroy/Gregoire's compiled reduction.

What is Normalization By Evaluation?



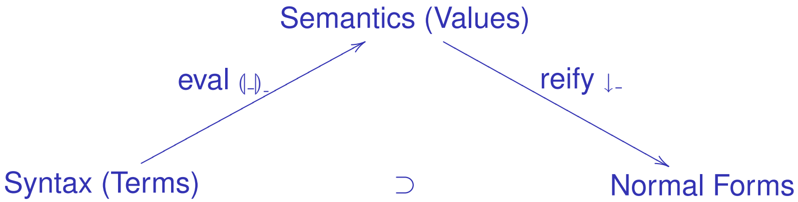
- You have: an interpreter ($\llparenthesis \_ \rrparenthesis_{\_}$).
- You buy: a reifyer ($\downarrow \_$).
- You get for free: a *full normalizer*!
- Famous instance: Leroy/Gregoire's compiled reduction.

What is Normalization By Evaluation?



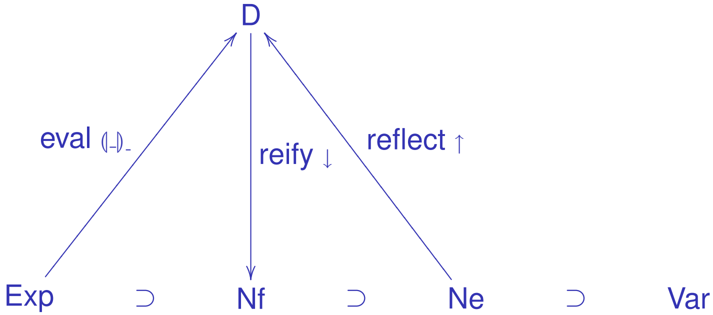
- You have: an interpreter ($\llparenthesis \_ \rrparenthesis_{\_}$).
- You buy: a reifyer ($\downarrow_{\_}$).
- You get for free: a *full normalizer*!
- Famous instance: Leroy/Gregoire's compiled reduction.

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.

Reflecting Variables into the Semantics



Interpretation in Scott Domain

- Set of values $\mathsf{D} \cong \mathsf{Ne} + [\mathsf{D} \to \mathsf{D}]$.
- Application operation $\_ \cdot \_ : \mathsf{D} \times \mathsf{D} \to \mathsf{D}$.
- Interpretation $(\!|t|\!)_\eta \in \mathsf{D}$ for term t and environment η:

$$
\begin{array}{rcl}
(\!|x|\!)_\eta & = & \eta(x) \\
(\!|r\,s|\!)_\eta & = & (\!|r|\!)_\eta \cdot (\!|s|\!)_\eta \\
(\!|\lambda x t|\!)_\eta \cdot d & = & (\!|t|\!)_{\eta[x \mapsto d]}
\end{array}
$$

Simply Typed Reification

- Reification D $\longrightarrow$ Nf:

$$\begin{array}{lcll} \downarrow^{S\to T} f & = & \lambda x\!:\!S.\ \downarrow^{T}(f\cdot(\uparrow^{S} x)) & \quad x \text{ fresh} \\ \downarrow^{*} u & = & u & \end{array}$$

- Reflection Ne $\longrightarrow$ D:

$$\begin{array}{lcl} (\uparrow^{S\to T} u)\cdot d & = & \uparrow^{T}(u\,(\downarrow^{S} d)) \\ \uparrow^{*} u & = & u \end{array}$$

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

Values in the CoC

- $\mathrm{Sort} \ni s ::= * \mid \Box$
- $\mathrm{D} \cong \mathrm{Ne} + \mathrm{Sort} + \mathrm{lam}^{\mathrm{Sort}}\, \mathrm{D}\, [\mathrm{D} \to \mathrm{D}] + \mathrm{fun}^{\mathrm{Sort}}\, \mathrm{D}\, [\mathrm{D} \to \mathrm{D}]$.
- Objects (programs)

$$\begin{array}{rcll} d, e, f & ::= & x\, \vec{v} & \text{neutral object} \\ & \mid & \mathrm{lam}^{\Box}\, K\, f & \text{type abstraction} \\ & \mid & \mathrm{lam}^{*}\, A\, f & \text{object abstraction} \end{array}$$

- Types

$$\begin{array}{rcll} A, B & ::= & X\, \vec{v} & \text{neutral type} \\ & \mid & \mathrm{fun}^{\Box}\, K\, F & \text{universal quantification} \\ & \mid & \mathrm{fun}^{*}\, A\, F & \text{dependent function type} \end{array}$$

Classification of Values (ctd.)

- Type constructors

$$
\begin{array}{rclll}
F, G & ::= & A & & \text{type} \\
& | & \mathrm{lam}^{\Box}\, K\, F & & \text{type abstraction} \\
& | & \mathrm{lam}^{*}\, A\, F & & \text{object abstraction}
\end{array}
$$

- Kinds (“types of type constructors”)

$$
\begin{array}{rclll}
K, L & ::= & * & & \\
& | & \mathrm{fun}^{\Box}\, K\, L & & \text{function kind} \\
& | & \mathrm{fun}^{*}\, A\, L & & \text{indexed kind}
\end{array}
$$

Reification and Reflection

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

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

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

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

Normalization algorithm

- $\mathsf{nbe}^T t = \downarrow^{(\!|T|\!)} (\!|t|\!)$
- Soundness: If $\vdash t : T$ then $\vdash t = \mathsf{nbe}^T t : T$.
- Completeness: If $\vdash t = t' : T$ then $\mathsf{nbe}^T t =_\alpha \mathsf{nbe}^T t'$.
- Show completeness using a PER model:
  1. If $\vdash t = t' : T$ then $((\!|t|\!), (\!|t'|\!)) \in [\![T]\!]$.
  2. If $\vdash T : *$ then $(\!|T|\!) \Vdash [\![T]\!]$.
  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 $\overline{*}$ 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 $[\![T]\!]$ of types.
6. Show the fundamental theorem:
   - $\vdash T : *$ implies $[\![T]\!]$ is a well-defined candidate.
   - $\vdash T = T' : *$ implies $[\![T]\!] = [\![T']\!]$.
   - $\vdash t : T$ implies $(\!|t|\!) \in [\![T]\!]$.

Interpretation of Types

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

Interpretation of Type Constructors

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

$$\begin{array}{lcl} \langle * \rangle & = & \mathrm{Per}(\mathrm{D}) \\ \langle k \to k' \rangle & = & \mathrm{D} \times \langle k \rangle \rightharpoonup \langle k' \rangle \\ \langle \diamond \to k \rangle & = & \mathrm{D} \rightharpoonup \langle k \rangle \end{array}$$

- A type constructor T of shape k is interpreted by a pair $(F, \mathcal{F})$
  1. $F \in \mathrm{D}$
  2. $\mathcal{F} \in \langle k \rangle$
  3. $F, F \Vdash^k \mathcal{F}$

$$\begin{array}{lcll} F, F' \Vdash^{k_1 \to k_2} \mathcal{F} & \text{iff} & F \cdot G, F' \cdot G \Vdash^{k_2} \mathcal{F}(G, \mathcal{G}) & \forall (G, \mathcal{G}) \in \mathrm{dom}(\mathcal{F}) \\ F, F' \Vdash^{\diamond \to k} \mathcal{F} & \text{iff} & F \cdot d, F' \cdot d \Vdash^{k} \mathcal{F}(d) & \forall d \in \mathrm{dom}(\mathcal{F}) \end{array}$$

- 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. $K \in \mathsf{D}$
  2. $\mathcal{K} \subseteq \mathsf{D} \times \mathsf{D} \times \langle k \rangle$
  3. $K \Vdash^{\square} \mathcal{K}$ meaning
     - $(\uparrow^K u, \uparrow^K u, \bot^k) \in \mathcal{K}$ for all $u \in \mathsf{Ne}$
     - $(F, F', \mathcal{F}) \in \mathcal{K}$ 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 $\forall\,\mathcal{K}\,\mathcal{F}$.

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

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

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

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: (partial) formalization in Coq.