Normalization by Evaluation for System F

Andreas Abel

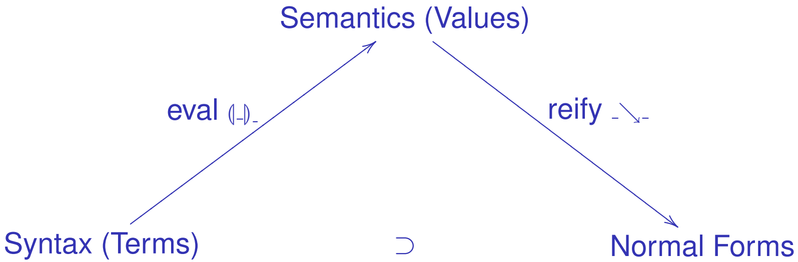
Department of Computer Science
Ludwig-Maximilians-University Munich

LPAR Conference, Doha, Qatar
26 November 2008

What is this for?

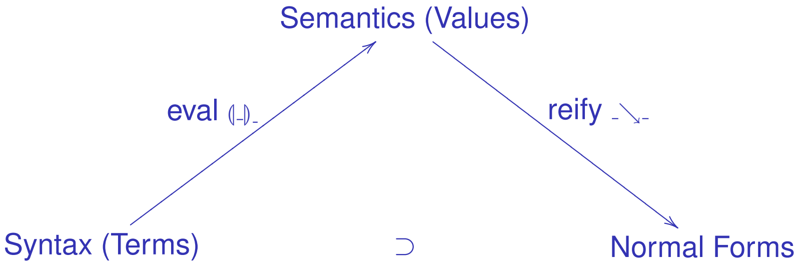
- Theorem provers based on Curry-Howard: Coq, Agda, ...
- Need to compare objects for equality.
- E.g. $f, g : \mathbb{N} \to \mathbb{N}$. Need a proof of $P(f)$, have one of $P(g)$.
- Extensional equality is undecidable.
- Approximation: intensional equality.
- Compute normal forms for f, g and compare.
- The more the better: β-, $\beta\eta$-, $\beta\eta\pi$-, $\ldots$ -normal form.
- NB: Coq distinguishes between $P(f)$ and $P(\lambda x.\, f\, x)$.
- Normalization-by-evaluation excellent when η is involved.

What is Normalization By Evaluation?



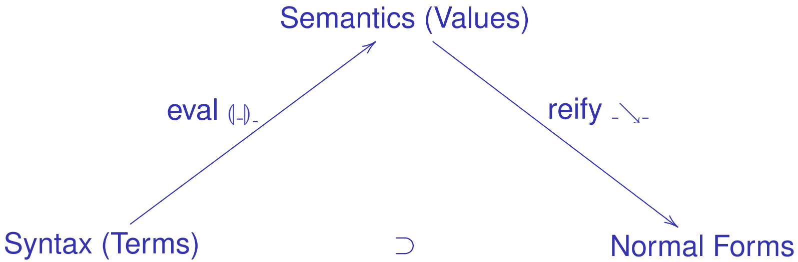
- You have: an interpreter (⦇_⦈_).
- You buy: my reifyer (_ ↘ _).
- You get for free: a *full normalizer*!

What is Normalization By Evaluation?



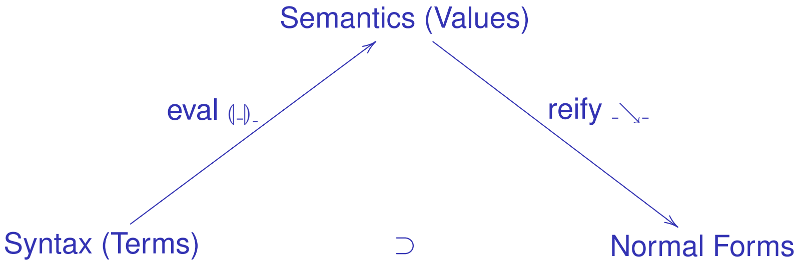
- You have: an interpreter ($(\!|\_|\!)_{\_}$).
- You buy: my reifyer ($\_ \searrow \_$).
- You get for free: a *full normalizer*!

What is Normalization By Evaluation?



- You have: an interpreter (⦇_⦈_).
- You buy: my reifyer (_ ↘ _).
- You get for free: a *full normalizer*!

What is Normalization By Evaluation?



- You have: an interpreter ($(\!|\_|\!)_{\_}$).
- You buy: my reifyer ($\_ \searrow \_$).
- You get for free: a *full normalizer*!

How to Reify a Function

- Functions are thought of as *black boxes*.
- How to print the code of a function?
- Apply it to a fresh variable!

$$\begin{array}{lcl} \mathsf{reify}\,(f) & = & \lambda x.\,\mathsf{reify}(f(x)) \\ \mathsf{reify}\,(x\,\vec{d}) & = & x\,\mathsf{reify}(\vec{d}) \end{array}$$

- Computation needs to be extended to handle variables (unknowns).

Choices of Semantics

1. β-normal forms (Agda 2, Ulf Norell)
2. Weak head normal forms (Constructive Engine, Randy Pollack)
3. Explicit substitutions (Twelf, Pfenning et.al.)
4. Closures (your favorite pure functional language, Epigram 2)
5. Virtual machine code (Coq: ZINC machine, Leroy/Gregoire)
6. Native machine code (Cayenne: i386, Dirk Kleeblatt)

These are all (partial) *applicative structures*.

Applicative Structures

An applicative structure consists of:

- A set D.
- Application operation $\_ \cdot \_ : \mathsf{D} \times \mathsf{D} \to \mathsf{D}$.
- Interpretation $(\!|t|\!)_\eta \in \mathsf{D}$ for term t and environment η, satisfying:

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

Simple examples:

1. $\mathsf{D} = (\mathsf{Tm}/{=_\beta})$ terms modulo β-equality.
2. $\mathsf{D} \cong [\mathsf{D} \to \mathsf{D}]$ reflexive (Scott) domain.

An Interpreter in Haskell

```
Abs :: (D -> D) -> D
app :: D -> (D -> D)

data Tm where
  TmVar :: Name -> Tm
  TmAbs :: Name -> Tm -> Tm
  TmApp :: Tm -> Tm -> Tm

lookup :: Env -> Name -> D
ext    :: Env -> Name -> D -> Env

eval :: Tm -> Env -> D
eval(TmVar x)  eta = lookup eta x
eval(TmAbs x t)eta = Abs (\ d -> eval t (ext eta x d))
eval(TmApp r s)eta = app (eval r eta) (eval s eta)
```

Applicative Structures with Variables

- Enrich D with all neutral objects $x\, d_1 \dots d_n$, where x a variable and $d_1, \dots, d_n \in \mathsf{D}$.
- Application satisfies:

$$(x\, \vec{d}) \cdot d = x\, \vec{d}\, d$$

- Leroy/Gregoire call neutral objects *accumulators*.

Value Domain with Variables

```
data D where
  Abs :: (D -> D) -> D
  Neu :: Ne -> D

type Name = String
data Ne where
  Var :: Name -> Ne
  App :: Ne -> D -> Ne

app :: D -> D -> D
app (Abs f) d  = f d
app (Neu n) d = Neu (App n d)
```

Reification (Simply-Typed)

- Given a type and a value of this type, produce a term.
- Context Γ records types of free variables.
- Inductively defined relation $\Gamma \vdash d \searrow v \Uparrow A$.
- "In context Γ, value d reifies to term v at type A."

$$\frac{\Gamma, x : A \vdash d \cdot x \searrow v \Uparrow B}{\Gamma \vdash d \searrow \lambda x v \Uparrow A \to B}$$

$$\frac{\Gamma \vdash d_i \searrow v_i \Uparrow A_i \text{ for all } i}{\Gamma \vdash x\, \vec{d} \searrow x\, \vec{v} \Uparrow *} \Gamma(x) = \vec{A} \to *$$

- Inputs: Γ, d, A
- Output: v (β-normal η-long).

Reification (Step by Step)

- Reifying neutral values step by step:

$$\Gamma \vdash e \searrow u \Downarrow A \qquad e \text{ reifies to } u\text{, inferring type } A.$$

- Inputs: Γ, e (neutral value).
- Outputs: u (neutral β-normal η-long), A.
- Rules:

$$\frac{}{\Gamma \vdash x \searrow x \Downarrow \Gamma(x)} \qquad \frac{\Gamma \vdash e \searrow u \Downarrow A \to B \qquad \Gamma \vdash d \searrow v \Uparrow A}{\Gamma \vdash e\,d \searrow u\,v \Downarrow B}$$

$$\frac{\Gamma \vdash e \searrow u \Downarrow *}{\Gamma \vdash e \searrow u \Uparrow *}$$

-

Type-Directed Reification in Haskell

```
reify  :: Cxt -> Ty -> D -> Tm
reify' :: Cxt -> Ne -> (Tm, Ty)

reify gamma (Arr a b) f = TmAbs x
  (reify gamma' b (app f (Neu (Var x))))
  where x      = freshName gamma
        gamma' = push gamma x a
reify gamma (Base _) (Neu n) = fst (reify' gamma n)

reify' gamma (Var x)   = (TmVar x, lookup gamma x)
reify' gamma (App n d) = (TmApp r s, b)
  where (r, Arr a b) = reify' gamma n
        s            = reify gamma a d
```

Normalization by Evaluation

- Compose evaluation with reification:

$$\text{nbe}_A(t) = \text{the } v \text{ with } \vdash (\!|t|\!)_{\rho_{\text{id}}} \searrow v \Uparrow A$$

- Completeness: NbE returns identical normal forms for all $\beta\eta$-equal terms of the same type.

  If $\Gamma \vdash t = t' : A$ *then* $\Gamma \vdash (\!|t|\!)_{\rho_{\text{id}}} \searrow v \Uparrow A$ *and* $\Gamma \vdash (\!|t'|\!)_{\rho_{\text{id}}} \searrow v \Uparrow A$.

- Soundness: NbE does not identify too many terms. The returned normal form is $\beta\eta$-equal to the original term.

  If $\Gamma \vdash t : A$ *then* $\Gamma \vdash (\!|t|\!)_{\rho_{\text{id}}} \searrow v \Uparrow A$ *and* $\Gamma \vdash t = v : A$.

- Both proven by Kripke logical relations.

A Logical Relation for Soundness

- A Kripke logical relation $\mathcal{A} \in \mathbb{K}^{A}$ of type A is a map from contexts Γ to relations between values and terms of type A:

$$(\Gamma \in \mathsf{Cxt}) \to \mathcal{P}(\mathsf{D} \times \mathsf{Tm}^{A}_{\Gamma})$$

- Monotonicity: extending Γ increases the relation.
- For each type A, define KLRs $\underline{A}, \overline{A}$ by

$$\begin{array}{lcl} \overline{A}_{\Gamma} & = & \{(d,t) \mid \Gamma \vdash d \searrow v \Uparrow A \text{ and } \Gamma \vdash t = v : A \text{ for some } v\} \\ \underline{A}_{\Gamma} & = & \{(e,t) \mid \Gamma \vdash e \searrow v \Downarrow A \text{ and } \Gamma \vdash t = v : A \text{ for some } v\} \end{array}$$

- Soundness: If $\Gamma \vdash t : A$ then $((\!|t|\!)_{\rho_{\mathsf{id}}}, t) \in \overline{A}_{\Gamma}$.
- Define KLR $[\![A]\!] \subseteq \overline{A}$ and show $((\!|t|\!)_{\rho_{\mathsf{id}}}, t) \in [\![A]\!]_{\Gamma}$ (fundamental theorem).

Interpretation Space

- Function space: given $\mathcal{A} \in \mathbb{K}^A$ and $\mathcal{B} \in \mathbb{K}^B$, define

$$
\begin{array}{rcl}
(\mathcal{A} \Rightarrow \mathcal{B})_\Gamma & = & \{(f, r) \in \mathsf{D} \times \mathsf{Tm}_\Gamma^{A \to B} \mid (f \cdot d, r\, s) \in \mathcal{B}_{\Gamma'} \\
& & \quad \text{if } \Gamma' \text{ extends } \Gamma \text{ and } (d, s) \in \mathcal{A}_{\Gamma'}\}
\end{array}
$$

- $\underline{A}, \overline{A}$ form an *interpretation space*, i. e.:

$$
\begin{array}{rcl}
\underline{*} & \subseteq & \overline{*} \\
\underline{A} \Rightarrow \overline{B} & \subseteq & \overline{A \to B} \\
\underline{A \to B} & \subseteq & \overline{A} \Rightarrow \underline{B}
\end{array}
$$

- We say $A \Vdash \mathcal{A}$ (A realizes $\mathcal{A}$) if $\underline{A} \subseteq \mathcal{A} \subseteq \overline{A}$.

Type interpretation

- Define $[\![A]\!]$ by induction on A.

$$
\begin{array}{lcl}
[\![*]\!] & = & \overline{*} \\
[\![A \to B]\!] & = & [\![A]\!] \Rightarrow [\![B]\!]
\end{array}
$$

- Theorem: $A \Vdash [\![A]\!]$.
- Now, the fundamental theorem implies soundness of NbE.
- Completeness by a similar logical relation.

What Have We Got?

- Abstractions in our proof:
  1. Applicative structures abstract over values and β.
  2. Fundamental theorem in a general form.
  3. Interpretation spaces abstract over “good” semantical types. *(New!)*
- Other instances for $\underline{A}$, $\overline{A}$ yield traditional weak $\beta(\eta)$-normalization.
- Readily adapts to System F.

Scaling to System F

- Extending the notion of interpretation space:

$$
\begin{array}{lcl}
(\bigcap_B \overline{A[B/Y]}) & \subseteq & \overline{\forall Y A} \\
\underline{\forall Y A} & \subseteq & \bigcap_B \underline{A[B/Y]}
\end{array}
$$

- Extending type interpretation:

$$
\begin{array}{rcl}
[\![X]\!]_\rho & = & \rho(X) \\
[\![A \to B]\!]_\rho & = & [\![A]\!]_\rho \to [\![B]\!]_\rho \\
[\![\forall X A]\!]_\rho & = & \bigcap_{B \Vdash \mathcal{B}} [\![A]\!]_{\rho[X \mapsto \mathcal{B}]}
\end{array}
$$

- Extending applicative structures, reification... (unproblematic).

Related Work

- Altenkirch, Hofmann, and Streicher (1997) describe another version of NbE for System F.
- Each type is interpreted by a syntactical type A, a semantical type $\mathcal{A}$, and a normalization function nf^A for terms of type A.
- Construction carried out in category theory.
- Other work on NbE: Schwichtenberg, Berger, Danvy, Filinski, Dybjer, Scott, Aehlig, Joachimski, Coquand, and many more.

Conclusions

- This work: NbE for System F with conventional means.
- Follows the structure of a weak normalization proof.
- Variation of Girard's scheme.
- Future work: scale to the Calculus of Constructions.

Acknowledgments: This work was carried out during a visit to Frédéric Blanqui and Cody Roux at LORIA, Nancy, France, financed by the Bayerisch-Französisches Hochschulzentrum.