

# Normalization by Evaluation for System F

Andreas Abel

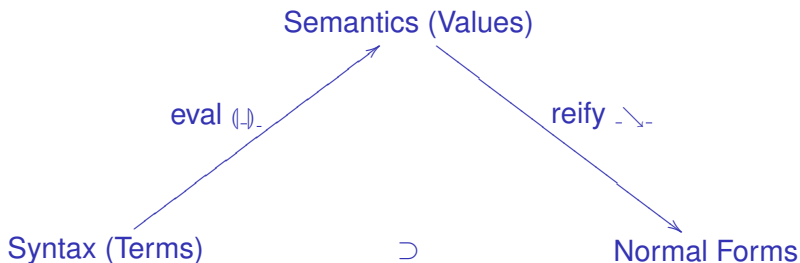
Department of Computer Science  
Ludwig-Maximilians-University Munich

National Institute for Informatics  
Tokyo, Japan  
5 December 2008

## What is this for?

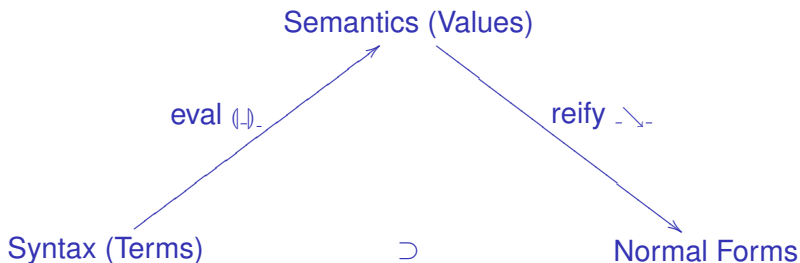
- Theorem provers based on Curry-Howard: Coq, Agda, ...
- Need to compare objects for equality.
- E.g.  $f, g : \mathbb{N} \rightarrow \mathbb{N}$ . Need a proof of  $P(f)$ , have one of  $P(g)$ .
- Extensional equality is undecidable.
- Approximation: intensional equality.
- Compute normal forms for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

# What is Normalization By Evaluation?



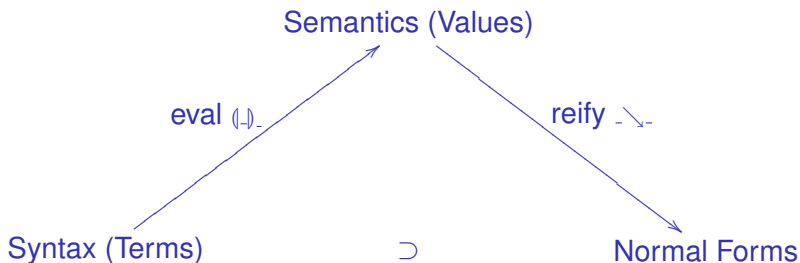
- You have: an interpreter `(|_)_`.
- You buy: my reifier `(- \to -)`.
- You get for free: a *full normalizer!*

# What is Normalization By Evaluation?



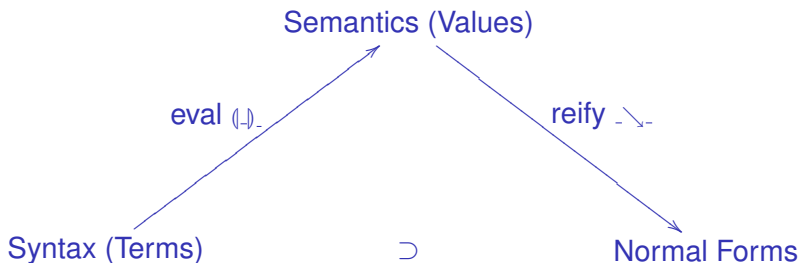
- You have: an interpreter  $(|-)$ .
- You buy: my reifier  $(- \rightarrow -)$ .
- You get for free: a *full normalizer*!

# What is Normalization By Evaluation?



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

# What is Normalization By Evaluation?



- You have: an interpreter  $(|-)$ .
- You buy: my reifyer  $(- \to -)$ .
- 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!

$$\begin{aligned}\text{reify}(f) &= \lambda x. \text{reify}(f(x)) \\ \text{reify}(x \vec{d}) &= x \text{reify}(\vec{d})\end{aligned}$$

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

## Choices of Semantics

- 1  $\beta$ -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  $\_ \cdot \_ : D \times D \rightarrow D$ .
- Interpretation  $\langle t \rangle_\eta \in D$  for term  $t$  and environment  $\eta$ , satisfying:

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

Simple examples:

- 1  $D = (\text{Tm}/\equiv_\beta)$  terms modulo  $\beta$ -equality.
- 2  $D \cong [D \rightarrow D]$  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  $d_1, \dots, d_n \in D$ .

- 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  $\Gamma$  records types of free variables.
- Inductively defined relation  $\Gamma \vdash d \searrow v \uparrow A$ .
- “In context  $\Gamma$ , value  $d$  reifies to term  $v$  at type  $A$ .”

$$\frac{\Gamma, x:A \vdash d \cdot x \searrow v \uparrow B}{\Gamma \vdash d \searrow \lambda xv \uparrow A \rightarrow B}$$

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

- Inputs:  $\Gamma, d, A$
- Output:  $v$  ( $\beta$ -normal  $\eta$ -long).

# Reification (Step by Step)

- Reifying neutral values step by step:

$\Gamma \vdash e \searrow u \Downarrow A$      $e$  reifies to  $u$ , inferring type  $A$ .

- Inputs:  $\Gamma$ ,  $e$  (neutral value).
- Outputs:  $u$  (neutral  $\beta$ -normal  $\eta$ -long),  $A$ .
- Rules:

$$\frac{}{\Gamma \vdash x \searrow x \Downarrow \Gamma(x)}$$
$$\frac{\Gamma \vdash e \searrow u \Downarrow A \rightarrow B \quad \Gamma \vdash d \searrow v \Uparrow A}{\Gamma \vdash ed \searrow uv \Downarrow B}$$
$$\frac{\Gamma \vdash e \searrow u \Downarrow *}{\Gamma \vdash e \searrow u \Uparrow *}$$



# 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.

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

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

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

- 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  $\Gamma$  to relations between values and terms of type  $A$ :

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

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

$$\begin{aligned}\bar{A}_\Gamma &= \{(d, t) \mid \Gamma \vdash d \searrow v \uparrow A \text{ and } \Gamma \vdash t = v : A \text{ for some } v\} \\ \underline{A}_\Gamma &= \{(e, t) \mid \Gamma \vdash e \searrow v \downarrow A \text{ and } \Gamma \vdash t = v : A \text{ for some } v\}\end{aligned}$$

- Soundness: If  $\Gamma \vdash t : A$  then  $((t)_{\rho_{\text{id}}}, t) \in \bar{A}_\Gamma$ .
- Define KLR  $\llbracket A \rrbracket \subseteq \bar{A}$  and show  $((t)_{\rho_{\text{id}}}, t) \in \llbracket A \rrbracket_\Gamma$  (fundamental theorem).

# Interpretation Space

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

$$(\mathcal{A} \Rightarrow \mathcal{B})_{\Gamma} = \{(f, r) \in D \times \text{Tm}_{\Gamma}^{A \rightarrow B} \mid (f \cdot d, r s) \in \mathcal{B}_{\Gamma'} \\ \text{if } \Gamma' \text{ extends } \Gamma \text{ and } (d, s) \in \mathcal{A}_{\Gamma'}\}$$

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

$$\begin{array}{ccc} * & \subseteq & * \\ \underline{A} \Rightarrow \overline{B} & \subseteq & \overline{A \rightarrow B} \\ \underline{A \rightarrow B} & \subseteq & \overline{A} \Rightarrow \underline{B} \end{array}$$

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

## Type interpretation

- Define  $\llbracket A \rrbracket$  by induction on  $A$ .

$$\begin{aligned}\llbracket * \rrbracket &= \bar{*} \\ \llbracket A \rightarrow B \rrbracket &= \llbracket A \rrbracket \Rightarrow \llbracket B \rrbracket\end{aligned}$$

- Theorem:  $A \Vdash \llbracket A \rrbracket$ .
- 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  $\beta$ .
  - 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:

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

- Extending type interpretation:

$$\begin{aligned}[[X]]_\rho &= \rho(X) \\ [[A \rightarrow B]]_\rho &= [[A]]_\rho \rightarrow [[B]]_\rho \\ [[\forall X A]]_\rho &= \bigcap_{B \Vdash B} [[A]]_{\rho[X \mapsto B]}\end{aligned}$$

- 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.*

*I thank the National Institute of Informatics and Makoto Tatsuta for the invitation.*