Normalization by Evaluation for Martin-Löf Type Theory

Andreas Abel[1] Klaus Aehlig[2]
Thierry Coquand[3] Peter Dybjer[3]

[1]Ludwig-Maximilians-University Munich
[2]University of Wales Swansea
[3]Chalmers University of Technology

LORIA, Nancy, France
February 27, 2007

Normalization by Evaluation

- Slogan: implement computation in the *object language (syntax)* by mapping it to computation in the *meta language (semantics)*
- For the λ-calculus: interpret object-level abstraction and application by meta-level abstraction and application
- Berger Schwichtenberg (LICS91): objects of the meta-language can be brought "down" to the object-language

Evaluation of λ-terms

- λ-calculus:

$$\mathsf{Tm} \ni r, s, t ::= x \mid \lambda x t \mid r\, s$$

- Applicative structure:

$$\begin{array}{lcll} \mathsf{D} & & & \text{some set} \\ \_ \cdot \_ & : & \mathsf{D} \times \mathsf{D} \to \mathsf{D} & \text{application} \\ [\![\_]\!]\_ & : & \mathsf{Tm} \times (\mathsf{Var} \to \mathsf{D}) \to \mathsf{D} & \text{evaluation} \end{array}$$

- Rules for evaluation and application:

$$\begin{array}{lcll} [\![x]\!]_\rho & = & \rho(x) & \text{variable} \\ [\![r\, s]\!]_\rho & = & [\![r]\!]_\rho \cdot [\![s]\!]_\rho & \text{application} \\ [\![\lambda x t]\!]_\rho \cdot d & = & [\![t]\!]_{\rho[x \mapsto d]} & \end{array}$$

Scott domain

- A specific D: Solution of domain equation

$$\mathsf{D} \cong [\mathsf{D} \to \mathsf{D}]$$

- The witnesses of the isomorphism

$$\begin{array}{lcl} \mathsf{Lam} & : & [\mathsf{D} \to \mathsf{D}] \to \mathsf{D} \\ \_ \cdot \_ & : & \mathsf{D} \to [\mathsf{D} \to \mathsf{D}] \\ (\mathsf{Lam}\, f) \cdot d & = & f(d) \end{array}$$

- Evaluation of abstraction:

$$[\![\lambda x t]\!]_\rho = \mathsf{Lam}\,(d \mapsto [\![t]\!]_{\rho[x \mapsto d]})$$

Implementation in Fresh Ocaml

```
type t;;        type var = t name;;
type term = Var of var
          | Lam of <<var>>term
          | App of term*term;;

type sem = LAM of (sem -> sem);;
let app e d = match e with LAM f -> f d;;

type env = var -> sem;;
let update rho x d = fun y -> if x=y then d else rho y;;

let rec eval t rho = match t with
    Var x           -> rho x
  | Lam (<<x>>t')   -> LAM (fun d -> eval t' (update rho x d))
  | App (r, s)      -> app (eval r rho) (eval s rho);;
```

Inverting evaluation

- Goal: A reification map $\downarrow : \mathsf{D} \to \mathsf{Nf}$.
- β-normal terms:

$$\begin{array}{llll} \mathsf{Nf} \ni v & ::= & \lambda x v \mid u & \text{normal term} \\ \mathsf{Ne} \ni u & ::= & x \mid u\, v & \text{neutral term} \end{array}$$

- Idea for $\downarrow$: Apply functions in D to fresh variables.
- First try:

$$\begin{array}{lcl} \mathsf{D} & \cong & [\mathsf{D} \to \mathsf{D}] + \mathsf{Var} \\ \mathsf{Up} & : & \mathsf{Var} \to \mathsf{D} \\ \\ \downarrow(\mathsf{Lam}\, f) & = & \lambda x.\, \downarrow(f\,(\mathsf{Up}\, x)) \\ \downarrow(\mathsf{Up}\, x) & = & x \end{array}$$

- But how to define application $(\mathsf{Up}\, x) \cdot d = ?$

Residualizing Semantics

- Solution for application:

$$\begin{array}{lcl} (\mathsf{Lam}\, f) \cdot d & = & f(d) \\ (\mathsf{Up}\, u) \cdot d & = & \mathsf{Up}\,(u\,(\downarrow d)) \end{array}$$

- Extend semantics to include neutral terms:

$$\begin{array}{lcl} \mathsf{D} & \cong & [\mathsf{D} \to \mathsf{D}] + \mathsf{Ne} \\ \mathsf{Up} & : & \mathsf{Ne} \to \mathsf{D} \\ \\ \downarrow(\mathsf{Lam}\, f) & = & \lambda x.\downarrow(f(\mathsf{Up}\, x)) \\ \downarrow(\mathsf{Up}\, u) & = & u \end{array}$$

Normalization by evaluation

- Normalization of closed terms:

$$\mathsf{nbe}\, t = \downarrow [\![t]\!]$$

- Completeness of NbE:

$$t =_\beta t' \implies \mathsf{nbe}\, t =_\alpha \mathsf{nbe}\, t'$$

- Soundness of NbE:

$$t =_\beta \mathsf{nbe}\, t$$

Implementation in Fresh Ocaml

```
type sem = LAM of (sem -> sem)
         | UP of term;;

let rec down d = match d with
    LAM f -> let x=fresh in Lam(<<x>>(down(f (UP (Var x)))))
  | UP t  -> t;;

let app e d = match e with
    LAM f -> f d
  | UP t  -> UP (App (t, down d));;

let norm t = down (eval t (fun x -> UP (Var x)));;
```

Simply-typed λ-calculus

- Simple types:

$$\mathsf{Ty} \ni A, B, C ::= o \mid A \to B$$

- η-long normal form: each $v : A \to B$ is a $\lambda x v'$.
- Reification is type-directed.

$$\begin{array}{lcl} \downarrow^{A \to B} e & = & \lambda x. \downarrow^{B} (e \cdot (\mathsf{Up}\, x)) \\ \downarrow^{o} (\mathsf{Up}\, u) & = & u \end{array}$$

- Example:

$$\begin{array}{lcl} \downarrow^{(o \to o) \to o \to o} [\![\lambda x x]\!] & = & \downarrow^{(o \to o) \to o \to o} (\mathsf{Lam}\, (d \mapsto d)) \\ & = & \lambda x. \downarrow^{o \to o} (\mathsf{Up}\, x) \\ & = & \lambda x \lambda y. \downarrow^{o} ((\mathsf{Up}\, x) \cdot (\mathsf{Up}\, y)) \\ & = & \lambda x \lambda y. \downarrow^{o} (\mathsf{Up}\, (x\, y)) \\ & = & \lambda x \lambda y. x\, y \end{array}$$

Simply-typed λ-calculus

- η-expand neutral terms in semantics.

$$\begin{array}{lcl} \uparrow^{o} u & = & \mathsf{Up}\, u \\ \uparrow^{A\to B} u & = & \mathsf{Lam}\,(d \mapsto \uparrow^{B}(u\,(\downarrow^{A} d))) \end{array}$$

$$\begin{array}{lcl} \downarrow^{o}(\mathsf{Up}\, u) & = & u \\ \downarrow^{A\to B} e & = & \lambda x.\downarrow^{B}(e \cdot (\uparrow^{A} x)) \end{array}$$

- Application: $(\mathsf{Lam}\, f) \cdot d = f(d)$.
- NbE produces η-long forms:

$$\begin{array}{lcl} \downarrow^{((o\to o)\to o)\to(o\to o)\to o}[\![\lambda x x]\!] & = & \downarrow^{((o\to o)\to o)\to(o\to o)\to o}(\mathsf{Lam}\,(d \mapsto d)) \\ & = & \lambda x.\downarrow^{(o\to o)\to o}(\uparrow^{(o\to o)\to o} x) \\ & = & \lambda x\lambda y.\downarrow^{o}(\uparrow^{(o\to o)\to o} x) \cdot (\uparrow^{o\to o} y) \\ & = & \lambda x\lambda y.\downarrow^{o}(\mathsf{Up}\,(x\downarrow^{o\to o}(\uparrow^{o\to o} y))) \\ & = & \lambda x\lambda y.\, x\,(\lambda z.\, y\, z) \end{array}$$

.

Martin Löf Type Theory

- Set Tm of raw terms:

$$r, s, t, A, B, C ::= x \mid \lambda x t \mid r\, s \mid \Pi x : A. B \mid \mathsf{Set}$$

- Plus natural numbers and primitive recursion like in Gödel's T
- Non-dependent function space $A \to B := \Pi \_ : A.\, B$
- $\beta\eta$-reduction: Congruence closure of

$$\begin{array}{lcll} (\lambda x t)\, s & \longrightarrow & t[s/x] & \\ \lambda x.\, (t\, x) & \longrightarrow & t & \text{if } x \notin \mathsf{FV}(t) \end{array}$$

- Plus computation rules for primitive recursion

Example: Functions with Variable Arity

- A dependent type

$$
\begin{array}{lcl}
T & : & \mathsf{Nat} \to \mathsf{Set} \\
T\,0 & = & \mathsf{Nat} \\
T\,(n+1) & = & \mathsf{Nat} \to T\,n \\
 & & \\
T\,3 & = & \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}
\end{array}
$$

- Summation of $n+1$ natural numbers

$$
\begin{array}{lcll}
sum' & : & \Pi n\!:\!\mathsf{Nat}.\,\mathsf{Nat} \to T\,n & \\
sum'\,0\,a & = & a & : \quad T\,0 \\
sum'\,(n+1)\,a\,x & = & sum'\,n\,(a+x) & : \quad T\,n \\
 & & & \\
sum'\,3\,0\,6\,7\,8 & = & 21 &
\end{array}
$$

Inference Rules

- Judgements

Γ cxt	Γ is a well-formed context
$\Gamma \vdash A$	A is a well-formed type
$\Gamma \vdash t : A$	t has type A

- Wellformed types

$$\frac{\Gamma \text{ cxt}}{\Gamma \vdash \mathsf{Set}} \qquad \frac{\Gamma \vdash A : \mathsf{Set}}{\Gamma \vdash A} \qquad \frac{\Gamma \vdash A \qquad \Gamma, x : A \vdash B}{\Gamma \vdash \Pi x : A.\, B}$$

Inference Rules II

- Lambda-calculus

$$\frac{\Gamma \text{ cxt}}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x t : \Pi x : A.\, B}$$

$$\frac{\Gamma \vdash r : \Pi x : A.\, B \qquad \Gamma \vdash s : A}{\Gamma \vdash r\, s : B[s/x]}$$

- Π in Set, conversion:

$$\frac{\Gamma \vdash A : \mathsf{Set} \qquad \Gamma, x : A \vdash B : \mathsf{Set}}{\Gamma \vdash \Pi x : A.\, B : \mathsf{Set}} \qquad \frac{\Gamma \vdash t : A \qquad \Gamma \vdash A'}{\Gamma \vdash t : A'}\ A =_{\beta\eta} A'$$

- Plus rules for natural numbers

Interpretation Domain

- Scott domain

$$\mathsf{D} \quad = \quad \mathsf{Lam}\,[\mathsf{D} \to \mathsf{D}] + \mathsf{Pi}\,\mathsf{D}\,[\mathsf{D} \to \mathsf{D}] + \mathsf{Set} + \mathsf{Up}\,\mathsf{Ne}$$

- Application

$$\begin{array}{lcll} \_ \cdot \_ & : & [\mathsf{D} \to [\mathsf{D} \to \mathsf{D}]] & \\ (\mathsf{Lam}\, f) \cdot d & = & f(d) & \\ e \cdot d & = & \bot & \text{if } e \text{ is not a } \mathsf{Lam} \end{array}$$

- "Bad guy" in D is, e. g., $\mathsf{Lam}\, f$ where

$$\begin{array}{lcl} f(\mathsf{Up}\, x) & = & \mathsf{Up}\, y \\ f(d) & = & \bot \end{array}$$

Evaluation

- Let $\rho \in \mathsf{Var} \to \mathsf{D}$ an environment.

$$\begin{aligned}
[\![x]\!]_\rho &= \rho(x) \\
[\![\lambda x t]\!]_\rho &= \mathsf{Lam}\,(d \mapsto [\![t]\!]_{\rho[x \mapsto d]}) \\
[\![r\ s]\!]_\rho &= [\![r]\!]_\rho \cdot [\![s]\!]_\rho \\
[\![\Pi x : A.\ B]\!]_\rho &= \mathsf{Pi}\,[\![A]\!]_\rho\,(d \mapsto [\![B]\!]_{\rho[x \mapsto d]}) \\
[\![\mathsf{Set}]\!]_\rho &= \mathsf{Set}
\end{aligned}$$

- $(\mathsf{D}, \_ \cdot \_, [\![\_]\!]_\_)$ is a weakly extensional λ-model

$$t =_\beta t' \quad \text{implies} \quad [\![t]\!]_\rho = [\![t']\!]_\rho$$

Reification and Reflection

- Bringing semantic objects down to $\beta\eta$-long forms

$$
\begin{array}{lcll}
\downarrow : [\mathsf{D} \times \mathsf{D} \to \mathsf{Nf}] & & & \\
\downarrow^{\mathsf{Set}} \mathsf{Pi}\, a\, g & = & \Pi x : \downarrow^{\mathsf{Set}} a.\, \downarrow^{\mathsf{Set}} g(\uparrow^{a} x) & \\
\downarrow^{\mathsf{Pi}\, a\, g} e & = & \lambda x.\, \downarrow^{g(\uparrow^{a} x)} (e \cdot (\uparrow^{a} x)) & \\
\downarrow^{c} (\mathsf{Up}\, t) & = & t & \text{if } c \text{ not a } \mathsf{Pi}
\end{array}
$$

- Embedding neutral terms $x\, t_1\, \ldots\, t_n$ into D

$$
\begin{array}{lcll}
\uparrow : [\mathsf{D} \times \mathsf{Ne} \to \mathsf{D}] & & & \\
\uparrow^{\mathsf{Pi}\, a\, g} t & = & \mathsf{Lam}\, (d \mapsto \uparrow^{g(d)} (t\, (\downarrow^{a} d))) & \\
\uparrow^{c} t & = & \mathsf{Up}\, t & \text{if } c \text{ not a } \mathsf{Pi}
\end{array}
$$

NbE Algorithm

- Environment ρ_Γ satisfies $(\rho_\Gamma)(x) = \uparrow^a x$ where $a = [\![\Gamma(x)]\!]_{\rho_\Gamma}$.
- A term $\Gamma \vdash t : A$ is normalized by

$$\mathsf{nbe}^A_\Gamma\, t = \downarrow^{[\![A]\!]_{\rho_\Gamma}} [\![t]\!]_{\rho_\Gamma}$$

Soundness and Completeness

- Goal: show that $\mathsf{nbe}^A_\Gamma\ t$ produces a canonical form for $\Gamma \vdash t : A$.
- Consequence: NbE decides $\beta\eta$-equality.
- Completeness: If $\Gamma \vdash t, t' : A$ then

$$t =_{\beta\eta} t' \text{ implies } \mathsf{nbe}^A_\Gamma\ t \equiv \mathsf{nbe}^A_\Gamma\ t'.$$

- Soundness: If $\Gamma \vdash t : A$ then

$$t =_{\beta\eta} \mathsf{nbe}^A_\Gamma\ t$$

Completeness

- Partial equivalence relation (PER) $c = c' \in \mathcal{T}ype$ identifies codes for semantical types $c, c' \in \mathsf{D}$.
- For $c = c \in \mathcal{T}ype$, define PER $e = e' \in [c]$.
- Definition mutually: use Dybjer's induction-recursion.

$$\frac{a = a' \in \mathcal{T}ype \qquad g(d) = g(d') \in \mathcal{T}ype \text{ for all } d = d' \in [a]}{\mathsf{Pi}\, a\, g = \mathsf{Pi}\, a'\, g' \in \mathcal{T}ype}$$

$$[\mathsf{Pi}\, a\, g] = \{(e, e') \mid e \cdot d = e' \cdot d' \in [g(d)] \text{ for all } d = d' \in [a]\}$$

- PERs single out total elements.
- PERs are extensional (handle η).

Completeness Proof (Sketch)

- Let $t, t' : A$.

$$t =_{\beta\eta} t'$$

$$\Downarrow$$

$$[\![t]\!] = [\![t']\!] \in [\![A]\!]$$

$$\Downarrow$$

$$\downarrow^{[\![A]\!]}[\![t]\!] \equiv \downarrow^{[\![A]\!]}[\![t']\!]$$

- Generalizes to open terms.

Soundness via Term Model

- For $\sigma \in \mathsf{Var} \to \mathsf{Tm}$ let $(\!|t|\!)_\sigma$ denote parallel substitution.
- $(\mathsf{Tm}/{=_{\beta\eta}}, \_, (\!|\_|\!)\_)$ is an extensional λ-model.
- Logical Relation $\mathsf{R}^a \subseteq (\mathsf{Tm}/{=_{\beta\eta}}) \times [a]$ for $a \in \mathcal{T}ype$.

$$
\begin{array}{lcll}
r\ \mathsf{R}^{\mathsf{Pi}\,a\,g}\ e & \iff & r\,s\ \mathsf{R}^{g(d)}\ e \cdot d & \text{for all } s\ \mathsf{R}^a\ d \\
r\ \mathsf{R}^c\ e & \iff & r =_{\beta\eta} \downarrow^c e &
\end{array}
$$

- Relating models: If $t : A$ then $(\!|t|\!)\ \mathsf{R}^{[\![A]\!]}\ [\![t]\!]$.
- Consequence $t =_{\beta\eta} \downarrow^{[\![A]\!]} (\!|t|\!)$.

Related Work

- Martin-Löf 1975: NbE for Type Theory (weak conversion)
- Martin-Löf 2004: Talk on NbE (philosophical justification)
- Danvy et al: Type-directed partial evaluation
- Altenkirch Hofmann Streicher 1996: NbE for λ-free System F
- Berger Eberl Schwichtenberg 2003: Term rewriting for NbE
- Aehlig Joachimski 2004: Untyped NbE, operationally
- Filinski Rohde 2004: Untyped NbE, denotationally
- Danielsson 2006: strongly typed NbE for LF