Normalization by Evaluation

Andreas Abel

Department of Computer Science and Engineering
Chalmers and Gothenburg University

PhD Seminar in Mathematical Engineering
EAFIT University, Medellin, Colombia
9 March 2017

Context of This Work

- Dependently-typed (programming) languages allow
  - to express functional specifications in types,
  - to prove (correctness) properties in the language,
  - formalize and prove mathematical propositions.
- Prominent proof assistent: Coq (INRIA 1984–)
  - CompCert: Certifed compiler for C– (Leroy)
  - Formalized proof of Four Color Theorem (Gonthier, 2005)
  - Odd-Order Theorem (Gonthier, 2012)

```
Theorem Feit_Thompson
  (gT : finGroupType) (G : {group gT}) :
    odd #|G| -> solvable G.
```

- Experimental languages: Agda, Idris, . . .

Behind the Veil

- What made Coq ready for huge developments?

  Benjamin Grégoire, Xavier Leroy:
  A compiled implementation of strong reduction. ICFP 2002
- Efficient normalization!
- Grégoire, Leroy: Efficient checking of β-equality.
- This talk: Framework for $\beta\eta$-equality.

A Taste of Programming with Dependent Types

- Descending lists: $[x, y, \ldots, z] \in \mathsf{List}^{\geq}\ n$ iff $n \geq x \geq y \geq \cdots \geq z$
- Constructor carries proof p for descent.

$$\frac{}{\mathsf{nil} : \mathsf{List}^{\geq}\ 0} \qquad \frac{x : \mathbb{N} \qquad p : x \geq y \qquad xs : \mathsf{List}^{\geq}\ y}{\mathsf{cons}\ x\ p\ xs : \mathsf{List}^{\geq}\ x}$$

- Singleton list carries a trivial proof.

$$\begin{array}{lcl} \mathsf{singleton} & : & (x : \mathbb{N}) \to \mathsf{List}^{\geq}\ x \\ \mathsf{singleton}\ x & = & \mathsf{cons}\ x\ \_\ \mathsf{nil} \ \ \text{where } \_ : x \geq 0 \end{array}$$

Correct Insert

- Case: Insert into empty list.

$$\begin{array}{lcl} \mathsf{insert} & : & (x : \mathbb{N}) \to \mathsf{List}^{\geq}\ n \to \mathsf{List}^{\geq}\ (\mathsf{max}\ x\ n) \\ \mathsf{insert}\ x\ \mathsf{nil} & = & \mathsf{singleton}\ x \end{array}$$

- Inferred type $\mathsf{singleton}\ x : \mathsf{List}^{\geq}\ x$.
- Expected type $\mathsf{singleton}\ x : \mathsf{List}^{\geq}\ (\mathsf{max}\ x\ 0)$.
- Type-checker needs to ensure $\mathsf{List}^{\geq}\ x = \mathsf{List}^{\geq}\ (\mathsf{max}\ x\ 0)$.
- Sufficient: $x = \mathsf{max}\ x\ 0$.
- Compare expressions with free variables!
- Solution: *normalize* $\mathsf{max}\ x\ 0$ to x.

Normalization

Bring an expression with unkowns into a canonical form.

- Unknowns = free variables.
- Checking equality by comparing canonical forms.
- Examples:

Expression	Normalizer
arithmetical expression	symbolic evaluator (MathLAB)
functional programming language	term rewriting, partial evaluation
stack maching code	JIT compiler
SQL query	query compiler

Evaluation

Compute the value of an expression relative to an environment.

- Environment assigns values to free variables of expressions.
- Examples:

Expression	Environment	Evaluator
arithmetical expression	valuation	calculator
functional programming language	stack & heap	interpreter
stack machine code	stack	stack machine
SQL-query	database	SQL processor

Normalization by Evaluation (NbE)

Adapt an interpreter to simplify expressions with unknowns.

- MLTT Martin-Löf 1975: NbE for Type Theory (weak conversion)
- STL Berger Schwichtenberg 1991: NbE for simply-typed λ-calculus
- T Danvy 1996: Type-directed partial evaluation
- F Altenkirch Hofmann Streicher 1996: NbE for λ-free System F
- λ Aehlig Joachimski 2004: Untyped NbE, operationally
- λ Filinski Rohde 2004: Untyped NbE, denotationally
- LF Danielsson 2006: strongly typed NbE for LF
- T Altenkirch Chapman 2007: Tait in one big step

Monoids

- Monoid $(M, \oplus, \varepsilon)$: set M with a binary operation $\oplus$ that has a unit ε.

$$
\begin{array}{rcll}
a \oplus (b \oplus c) & = & (a \oplus b) \oplus c & \text{associativity} \\
\varepsilon \oplus a & = & a & \text{left unit} \\
a \oplus \varepsilon & = & a & \text{right unit}
\end{array}
$$

- E.g.: $(\mathbb{N}, +, 0)$, $(\mathbb{N}, \cdot, 1)$, $(\mathsf{Bool}, \wedge, \mathsf{true})$, $(\mathsf{Bool}, \vee, \mathsf{false})$, $(R^{n \times n}, \cdot, I_n)$.
- Free monoid: Sequences with concatenation $(\mathsf{List}\, A, +\!\!+, [\,])$.

$$[a_1, \dots, a_m] \;+\!\!+\; [a_{m+1}, \dots, a_n] \;=\; [a_1, \dots, a_n]$$

Monoid Expressions

- Fix a carrier A and a set of variables X.
- Terms (abstract syntax trees) representing monoid elements:

$$
\begin{array}{rcll}
\mathrm{Exp} \ni t & ::= & a & \text{singleton } a \in A \\
 & \mid & x & \text{variable } x \in X \\
 & \mid & \varepsilon & \text{empty sequence} \\
 & \mid & t_1 \cdot t_2 & \text{concatenation, right associative}
\end{array}
$$

- Example in concrete syntax:

$$\mathrm{ex} \; := \; (x_0 \cdot 1) \cdot (((2 \cdot (\varepsilon \cdot x_1)) \cdot (\varepsilon \cdot 4)) \cdot x_2)$$

Interpreting Monoid Expressions

- Monoid values $\mathsf{Val} = \mathsf{List}\, A$.
- Environment $\rho \in X \to \mathit{Val}$.
- Interpretation $(\!| t |\!)_\rho \in \mathsf{Val}$.

$$
\begin{array}{lcll}
(\!| x |\!)_\rho & = & \rho(x) & \\
(\!| \varepsilon |\!)_\rho & = & [\,] & \text{empty list} \\
(\!| a |\!)_\rho & = & [a] & \text{singleton list} \\
(\!| t_1 \cdot t_2 |\!)_\rho & = & (\!| t_1 |\!)_\rho +\!\!+ (\!| t_2 |\!)_\rho & \text{append}
\end{array}
$$

- Example. Recall $\mathsf{ex} = (x_0 \cdot 1) \cdot (((2 \cdot (\varepsilon \cdot x_1)) \cdot (\varepsilon \cdot 4)) \cdot x_2)$

$$(\!| \mathsf{ex} |\!)_{(x_0=0, x_1=3, x_2=5)} = [0, 1, 2, 3, 4, 5]$$

Normalizing Monoid Expressions I

- $Val = \mathsf{List}\,(A \uplus X)$.
- Reflection of variables into values.

$$\begin{aligned} \uparrow \quad &: \quad X \to \mathsf{Val} \\ \uparrow x \quad &= \quad [x] \end{aligned}$$

- Reification of values as expressions.

$$\begin{aligned} \downarrow \quad &: \quad \mathsf{Val} \to \mathsf{Exp} \\ \downarrow [] \quad &= \quad \varepsilon \\ \downarrow (a :: v) \quad &= \quad a \cdot \downarrow v \\ \downarrow (x :: v) \quad &= \quad x \cdot \downarrow v \end{aligned}$$

Normalizing Monoid Expressions II

- Normalization:

$$\begin{array}{rcl} \mathsf{nf} & : & \mathsf{Exp} \to \mathit{Exp} \\ \mathsf{nf}(t) & = & \downarrow (\!| t |\!)_{\uparrow} \end{array}$$

- Example. Recall $\mathsf{ex} = (x_0 \cdot 1) \cdot (((2 \cdot (\varepsilon \cdot x_1)) \cdot (\varepsilon \cdot 4)) \cdot x_2)$

$$\mathsf{nf}(\mathsf{ex}) = x_0 \cdot 1 \cdot 2 \cdot x_1 \cdot 4 \cdot x_2 \cdot \varepsilon$$

Untyped Lambda Calculus, Informally

- Calculus of functions. Everything is a function.
- Examples:

id	$=$	$\lambda x.\, x$	identity function
app	$=$	$\lambda f.\, \lambda x.\, f\, x$	application function (also identity)
twice	$=$	$\lambda f.\, \lambda x.\, f\,(f\, x)$	apply f twice
comp	$=$	$\lambda f.\, \lambda g.\, \lambda x.\, f\,(g\, x)$	compose two functions

- Calculation:

$$\mathsf{app}\,\mathsf{twice}\,\mathsf{id} = \mathsf{twice}\,\mathsf{id} = \lambda x.\, \mathsf{id}\,(\mathsf{id}\, x) = \lambda x.\, \mathsf{id}\, x = \lambda x.\, x = \mathsf{id}.$$

Numbers in the Untyped Lambda Calculus

- Numbers $n \in \mathbb{N}$ are represented by Church numerals $\underline{n}$.

$$\begin{aligned}\underline{0} &= \lambda f.\, \lambda x.\, x \\ \underline{1} &= \lambda f.\, \lambda x.\, f\, x \\ \underline{2} &= \lambda f.\, \lambda x.\, f\, (f\, x) \\ \underline{n} &= \lambda f.\, \lambda x.\, f^n\, x\end{aligned}$$

- Addition is a sort of composition.

$$\mathsf{plus} = \lambda n\, m\, f\, x.\, n\, f\, (m\, f\, x)$$

- $\mathsf{plus}\, \underline{n}\, \underline{m} = \lambda f\, x.\, \underline{n}\, f\, (\underline{m}\, f\, x) = \lambda f\, x.\, f^n\, (f^m\, x) = \lambda f\, x.\, f^{n+m}\, x = \underline{n+m}$

Recursion in the Untyped Lambda Calculus

- Reduction:

$$(\lambda x.\, t)\, s \longrightarrow t[s/x]$$

- A looping term:

$$(\lambda x.\, x\, x)\, (\lambda x.\, x\, x) \longrightarrow (x\, x)[(\lambda x.\, x\, x)/x] = (\lambda x.\, x\, x)\, (\lambda x.\, x\, x)$$

- Alan Turing's fixed-point combinator. Let $\theta = (\lambda x.\, \lambda f.\, f\, (x\, x\, f))$.

$$\theta\, \theta\, f \longrightarrow f\, (\theta\, \theta\, f)$$

Untyped Lambda Calculus, Formally

- Grammar:

$$
\begin{array}{lllll}
\mathsf{Exp} & \ni & r, s, t & ::= & x & \text{variable} \\
 & & & \mid & \lambda x.\, t & \text{abstracting variable } x \text{ in body } t \\
 & & & \mid & r\, s & \text{applying } r \text{ to } s
\end{array}
$$

- Equational theory (β):

$$\vdash (\lambda x.\, t)\, s = t[s/x]$$

- β-normal forms.

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

Evaluation of Lambda-Expressions

- Values $a, b, f \in \mathsf{D}$ with (partial) application $\_ \cdot \_ : \mathsf{D} \times \mathsf{D} \to \mathsf{D}$.
- Evaluation (specification):

$$
\begin{array}{rcl}
(\!|x|\!)_\rho & = & \rho(x) \\
(\!|r\,s|\!)_\rho & \doteq & (\!|r|\!)_\rho \cdot (\!|s|\!)_\rho \\
(\!|\lambda x.\,t|\!)_\rho \cdot a & \doteq & (\!|t|\!)_{(\rho, a/x)}
\end{array}
$$

- Instance: compiled execution.

$f \cdot a$	Call f with argument a
$(\!|\lambda x.\,t|\!)_\rho$	Code for function $\lambda x.\,t$ with predefined variables ρ

Implementation via Closures

- Instance: do nothing.

$$(\!|\lambda x.\, t|\!)_\rho \quad = \quad (\underline{\lambda} x t)\rho$$

- Initial applicative structure: closures.

$$\mathsf{D} \quad \ni \quad a, b, f \quad ::= \quad (\underline{\lambda} x t)\rho \quad \text{waiting for value of } x$$

- Application and evaluation are mutually defined.

$$(\underline{\lambda} x t)\rho \cdot a \quad = \quad (\!|t|\!)_{(\rho, a/x)}$$

$$(\!|r\, s|\!)_\rho \quad = \quad (\!|r|\!)_\rho \cdot (\!|s|\!)_\rho$$

Residual Model: Adding Unknowns

- For normalization, we need free variables in D.
- Application $x \cdot a$ of a free variable stores argument a.
- Need neutrals/accumulators $x\,\vec{a}$ in D.

$$\begin{array}{lcccl} \mathsf{D} & \ni & a, b, f & ::= & (\underline{\lambda} x t)\rho \mid e \\ \mathsf{D}^{\mathsf{ne}} & \ni & e & ::= & x \mid e\,a \end{array}$$

- Application extended:

$$\begin{array}{lcl} (\underline{\lambda} x t)\rho \cdot a & = & (\!| t |\!)_{(\rho, a/x)} \\ x\,\vec{a} \cdot a & = & x\,(\vec{a}, a) \end{array}$$

Reading Back Expressions from Values

- Reading back values:

$$
\begin{array}{lcl}
\mathsf{R}^{\mathsf{nf}} & : & \mathsf{D} \to \mathsf{Nf} \\
\mathsf{R}^{\mathsf{nf}}((\underline{\lambda} x t)\rho) & = & \lambda y.\, \mathsf{R}^{\mathsf{nf}}(⦇t⦈_{(\rho, y/x)}) \quad \text{where } y \text{ “fresh”} \\
\mathsf{R}^{\mathsf{nf}}(e) & = & \mathsf{R}^{\mathsf{ne}}(e)
\end{array}
$$

- Reading back neutrals:

$$
\begin{array}{lcl}
\mathsf{R}^{\mathsf{ne}} & : & \mathsf{D}^{\mathsf{ne}} \to \mathsf{Ne} \\
\mathsf{R}^{\mathsf{ne}}(x) & = & x \\
\mathsf{R}^{\mathsf{ne}}(e\, a) & = & \mathsf{R}^{\mathsf{ne}}(e)\, \mathsf{R}^{\mathsf{nf}}(a)
\end{array}
$$

Fresh Name Generation

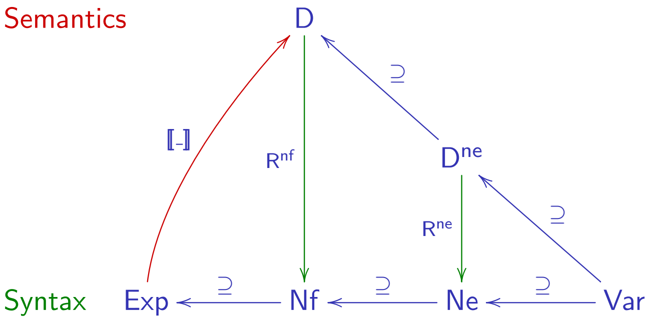
- Freshness problem: ≥ 9 approaches.
- Simple solution: $\mathsf{R}^{\mathsf{nf}}_{\xi}$ reads fresh names from supply ξ.
- E.g., ξ is an infinite stream of distinct identifiers.

$$
\begin{array}{lll}
\mathsf{R}^{\mathsf{nf}}_{(y,\xi)}((\underline{\lambda} x t)\rho) & = & \lambda y.\, \mathsf{R}^{\mathsf{nf}}_{\xi}(\llbracket t \rrbracket_{(\rho, y/x)}) \\
\mathsf{R}^{\mathsf{nf}}_{\xi} \quad (e) & = & \mathsf{R}^{\mathsf{ne}}_{\xi}(e) \\
\mathsf{R}^{\mathsf{ne}}_{\xi} \quad (x\, \vec{a}) & = & x\, \mathsf{R}^{\mathsf{nf}}_{\xi}(\vec{a})
\end{array}
$$

- Normalization:

$$
\mathsf{nf}_{\xi}(t) \quad = \quad \mathsf{R}^{\mathsf{nf}}_{\xi}(\llbracket t \rrbracket_{\rho_{\mathsf{id}}})
$$

Summary: NbE for Untyped Lambda-Calculus



Simply-Typed Lambda Calculus

- Types $S, T ::= \mathsf{N} \mid S \to T$.
- Typing contexts $\Gamma ::= x_1 : S_1, \ldots, x_n : \mathcal{S}_n$.
- Typing $\Gamma \vdash t : T$.

$$\frac{(x : T) \in \Gamma}{\Gamma \vdash x : T} \qquad \frac{\Gamma, x : S \vdash t : T}{\Gamma \vdash \lambda x.\, t : S \to T} \qquad \frac{\Gamma \vdash r : S \to T \qquad \Gamma \vdash s : S}{\Gamma \vdash r\, s : T}$$

- Equational theory $(\beta\eta)$.

$$(\beta)\ \frac{\Gamma, x : S \vdash t : T \qquad \Gamma \vdash s : S}{\Gamma \vdash (\lambda x t)\, s = t[s/x] : T}$$

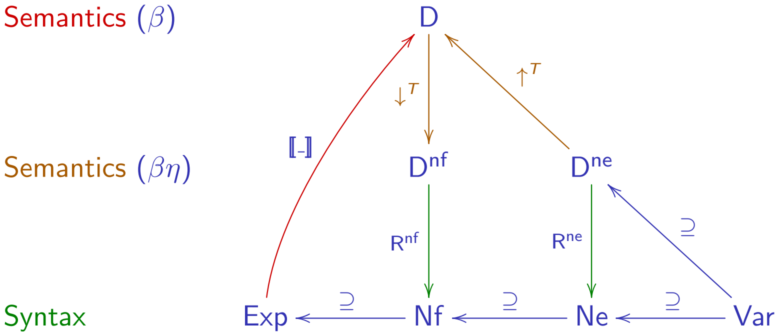
$$(\eta)\ \frac{\Gamma \vdash t : S \to T}{\Gamma \vdash t = \lambda x.\, t\, x : S \to T}$$

Bidirectional η-Expansion

- $\uparrow^T$ "reflection": η-expansion inside-out
- $\downarrow^T$ "reification": η-expansion outside-in
- Example (terms):

$$\begin{aligned}
\downarrow^{(\mathsf{N}\to\mathsf{N})\to(\mathsf{N}\to\mathsf{N})} f &= \lambda y.\, \downarrow^{\mathsf{N}\to\mathsf{N}}(f\ (\uparrow^{\mathsf{N}\to\mathsf{N}} y)) \\
&= \lambda y.\, \lambda x.\, \downarrow^{\mathsf{N}}(f\ (\uparrow^{\mathsf{N}\to\mathsf{N}} y)\ (\uparrow^{\mathsf{N}} x)) \\
&= \lambda y.\, \lambda x.\, \downarrow^{\mathsf{N}}(f\ (\lambda z.\, \uparrow^{\mathsf{N}}(y\ (\downarrow^{\mathsf{N}} z)))\ (\uparrow^{\mathsf{N}} x)) \\
&= \lambda y.\, \lambda x.\, f\ (\lambda z.\, y\ z)\ x
\end{aligned}$$

Adding η-Expansion



Eta-expansion: reflection and reification

- Values now include delayed η-expansions.

$$
\begin{array}{llll}
\mathsf{D} & \ni & a, b, f & ::= & (\underline{\lambda} x t)\rho \mid \uparrow^{T} e \\
\mathsf{D}^{\mathsf{ne}} & \ni & e & ::= & x \mid e\, d \\
\mathsf{D}^{\mathsf{nf}} & \ni & d & ::= & \downarrow^{T} a
\end{array}
$$

- Application and readback trigger these expansions.

$$
\begin{array}{lll}
(\underline{\lambda} x t)\rho \cdot a & = & (\!| t |\!)_{(\rho, a/x)} \\
\uparrow^{S \to T} e \cdot a & = & \uparrow^{T} (e \downarrow^{S} a)
\end{array}
$$

$$
\begin{array}{llll}
\mathsf{R}^{\mathsf{nf}}_{(y,\xi)} & (\downarrow^{S \to T} f) & = & \lambda y.\, \mathsf{R}^{\mathsf{nf}}_{\xi} (\downarrow^{T} (f \cdot \uparrow^{S} y)) \\
\mathsf{R}^{\mathsf{nf}}_{\xi} & (\downarrow^{\mathsf{N}} \uparrow^{\mathsf{N}} e) & = & \mathsf{R}^{\mathsf{ne}}_{\xi}(e)
\end{array}
$$

Normalization for STL

- Canonical environment:

$$\rho_\Gamma(x) = \uparrow^T x \text{ where } (x : T) \in \Gamma$$

- Variable supply:

$$\xi_\Gamma = \mathsf{Var} \setminus \Gamma$$

- Normalization of $\Gamma \vdash t : T$:

$$\mathsf{nf}^T_\Gamma(t) = \mathsf{R}^{\mathsf{nf}}_{\xi_\Gamma}(\downarrow^T (\!|t|\!)_{\rho_\Gamma})$$

Correctness of Normalization

- Normalization is sound if for all expressions $\Gamma \vdash t : T$,

$$\Gamma \vdash t = \mathsf{nf}_\Gamma^T(t) : T.$$

- Normalization is complete if for all $\Gamma \vdash t, t' : T$,

$$\Gamma \vdash t = t' : T \implies \mathsf{nf}_\Gamma^T(t) =_\alpha \mathsf{nf}_\Gamma^T(t')$$

- Implies idempotence $\mathsf{nf}_\Gamma^T(t) =_\alpha \mathsf{nf}_\Gamma^T(\mathsf{nf}_\Gamma^T(t))$.

Completeness of Normalization

Well-typed $\beta\eta$-equal terms have the same normal form.

$$\Gamma \vdash t = t' : T \quad \Longrightarrow \quad \overbrace{(\!|t|\!)_{\rho_\Gamma}}^{a} = \overbrace{(\!|t'|\!)_{\rho_\Gamma}}^{a'} \in \overbrace{(\!|T|\!)_{\rho_\Gamma}}^{A}$$

$$\Longrightarrow \quad a = a' \in \overline{A}$$

$$\Longrightarrow \quad \downarrow^{A} a = \downarrow^{A} a' \in \top$$

$$\Longrightarrow \quad \mathsf{R}^{\mathsf{nf}}_{\xi_\Gamma} \downarrow^{A} a =_{\alpha} \mathsf{R}^{\mathsf{nf}}_{\xi_\Gamma} \downarrow^{A} a'$$

Soundness of Normalization

A well-typed term is $\beta\eta$-equal to its normal form.

$$\Gamma \vdash t : T \quad \Longrightarrow \quad \Gamma \vdash t : T \;®\; \overbrace{(\!| t |\!)_{\rho_\Gamma}}^{a} \in \overbrace{(\!| T |\!)_{\rho_\Gamma}}^{A}$$

$$\Longrightarrow \quad \Gamma \vdash t = \mathsf{R}^{\mathsf{nf}}_{\xi_\Gamma} {\downarrow}^{A} a : T$$

$$\Longleftrightarrow \quad \Gamma \vdash t = \mathsf{nf}^{T}_{\Gamma}(t) : T$$

Conclusions

- Interpreters can be turned into normalizers in a systematic way.
- Normalization-by-evaluation has helped to understand η-equality.
- NbE is also a theoretical tool to investigate Type Theory.
- E.g., to prove decidability of type checking.