Normalization by Evaluation
Dependent Types and Impredicativity

Andreas Abel

Department of Computer Science
Ludwig-Maximilians-University Munich

Habilitation Talk
Ludwig-Maximilians-University Munich
31 May 2013

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, Epigram, 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 thesis: Framework for $\beta\eta$-equality.

A Taste of Programming with Dependent Types

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

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

- Singleton list carries a trivial proof.

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

Correct Insert

- Case: Insert into empty list.

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

- Inferred type $\mathsf{singleton}\ x : \mathsf{List}^{\downarrow}\ x$.
- Expected type $\mathsf{singleton}\ x : \mathsf{List}^{\downarrow}\ (\mathsf{max}\ x\ 0)$.
- Type-checker needs to ensure $\mathsf{List}^{\downarrow}\ x = \mathsf{List}^{\downarrow}\ (\mathsf{max}\ x\ 0)$.
- 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

This Thesis

A correct normalization-by-evaluation procedure for functional languages with dependent types and impredicative polymorphism.

- Start from untyped NbE.
- Semantics/model based on NbE.
- Model proves decidability of equality and typing.
- Model uses generic partial applicative structures.
- Covers many different implementations (e.g., Coq's compiled reduction).

Publications Underlying This Thesis

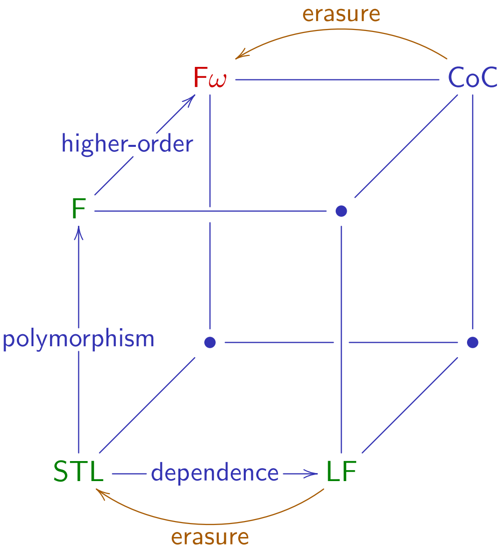
Dependent types:

- MLTT Abel Aehlig Dybjer (MFPS 2007): untyped equality.
- MLTT Abel Coquand Dybjer (LICS 2007):
  Decidability of typed equality with η on types.
- MLTT Abel Coquand Pagano (LMCS 2011): Singleton types.

Impredicativity:

- F Abel (LPAR 2008)
- $\mathsf{F}\omega$ Abel (CSL 2009)
- CoC+$\mathbb{N}$ Abel (FLOPS 2010)

Barendregt's λ-Cube



NbE:

known (STL, LF, F)

this thesis (Fω)

reducible (CoC)

Dependency Erasure

- Restrictive dependencies can be erased safely.

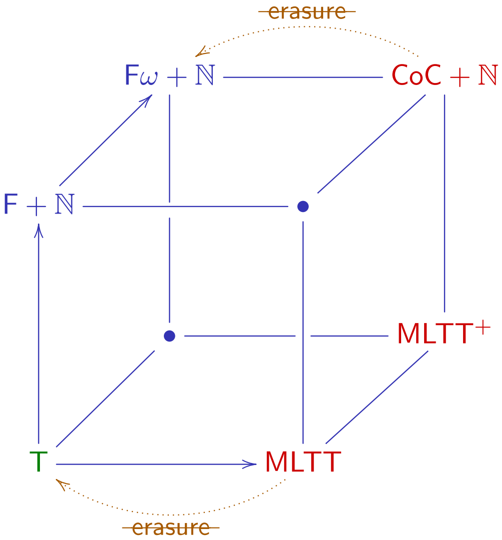
$$|\mathsf{List}^{\downarrow}\ n| = \mathsf{List}$$

- We can forget that we deal with descending lists.
- Recursive (computational) dependencies cannot be erased.

$$\begin{array}{lcl} \mathsf{NAry} & : & \mathbb{N} \to \mathit{Type} \\ \mathsf{NAry}\ 0 & = & \mathbb{N} \\ \mathsf{NAry}\ (n+1) & = & \mathbb{N} \to \mathsf{NAry}\ n \\ \\ |\mathsf{NAry}\ n| & = & ? \end{array}$$

- No simple type corresponds to $\mathsf{NAry}\ n$.

The Cube with $\mathbb{N}$ and Recursion



NbE:
known (T)
th. ($\mathsf{MLTT}(^{+}), \mathsf{CoC} + \mathbb{N}$)
subsumed ($F(\omega) + \mathbb{N}$)

Untyped Lambda Calculus

- Grammar:

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

- 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.

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

- Application and evaluation are mutually defined.

$$\begin{aligned} (\underline{\lambda} x t)\rho \cdot a &\quad = \quad (\!|t|\!)_{(\rho, a/x)} \\ (\!|r\, s|\!)_\rho &\quad = \quad (\!|r|\!)_\rho \cdot (\!|s|\!)_\rho \end{aligned}$$

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.

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

- Application extended:

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

Reading Back Expressions from Values

- Reading back values:

$$\begin{array}{lcl} \mathsf{R}^{\mathsf{nf}} & : & \mathsf{D} \to \mathsf{Nf} \\ \mathsf{R}^{\mathsf{nf}}((\underline{\lambda} x t)\rho) & = & \lambda y.\, \mathsf{R}^{\mathsf{nf}}(\llparenthesis t \rrparenthesis_{(\rho, y/x)}) \quad \text{where } y \text{ ``fresh''} \\ \mathsf{R}^{\mathsf{nf}}(e) & = & \mathsf{R}^{\mathsf{ne}}(e) \end{array}$$

- 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.

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

- Normalization:

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

Summ~~it~~ary

$$\begin{tikzcd}
\text{Semantics} & & D \arrow[dd, "R^{nf}"'] & & \\
 & & & D^{ne} \arrow[ul, "\supseteq"'] \arrow[d, "R^{ne}"'] & \\
\text{Syntax} & \mathsf{Exp} \arrow[uur, bend left, "\llbracket _ \rrbracket"] & \mathsf{Nf} \arrow[l, "\supseteq"'] & \mathsf{Ne} \arrow[l, "\supseteq"'] & \mathsf{Var} \arrow[l, "\supseteq"'] \arrow[ul, "\supseteq"']
\end{tikzcd}$$

Simply-Typed Lambda Calculus

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

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

Adding η-Expansion

$$
\begin{tikzcd}
\text{Semantics } (\beta) & & \mathsf{D} \arrow[d, "\downarrow^{T}"'] & & \\
\text{Semantics } (\beta\eta) & & \mathsf{D}^{\mathsf{nf}} \arrow[d, "\mathsf{R}^{\mathsf{nf}}"'] & \mathsf{D}^{\mathsf{ne}} \arrow[ul, "\uparrow^{T}"'] \arrow[d, "\mathsf{R}^{\mathsf{ne}}"'] & \\
\text{Syntax} & \mathsf{Exp} \arrow[uur, bend left, "[\![\_]\!]"] & \mathsf{Nf} \arrow[l, "\supseteq"'] & \mathsf{Ne} \arrow[l, "\supseteq"'] & \mathsf{Var} \arrow[l, "\supseteq"'] \arrow[ul, "\supseteq"']
\end{tikzcd}
$$

Eta-expansion: reflection and reification

- Values now include delayed η-expansions.

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

- Application and readback trigger these expansions.

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

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

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 $\Gamma \vdash t : T$:

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

Dependent Types

- Recall singleton : $(x : \mathbb{N}) \to \mathsf{List}^{\downarrow}\ x$.
- Dependent function space $(x : S) \to T$.

$$\frac{\Gamma \vdash S : \mathsf{Type} \qquad \Gamma, x\!:\!S \vdash T : \mathsf{Type}}{\Gamma \vdash (x\!:\!S) \to T : \mathsf{Type}}$$

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

- η-expansion directed by type values.

$$\begin{aligned}
\downarrow^{\mathsf{NAry}(2)}(f) &= \downarrow^{\mathsf{N} \to \mathsf{NAry}(1)}(f) \\
&= \lambda x.\, \downarrow^{\mathsf{NAry}(1)}(f\ (\uparrow^{\mathsf{N}} x)) \\
&= \lambda x.\, \downarrow^{\mathsf{N} \to \mathsf{NAry}(0)}(f\ (\uparrow^{\mathsf{N}} x)) \\
&= \lambda x.\, \lambda y.\, \downarrow^{\mathsf{NAry}(0)}(f\ (\uparrow^{\mathsf{N}} x)\ (\uparrow^{\mathsf{N}} y)) = \ldots
\end{aligned}$$

Type Values

- Values include types.

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

- Read-back evaluates types further.

$$
\begin{array}{lcl}
\mathsf{R}^{\mathsf{nf}}_{(y,\xi)}(\downarrow^{\mathsf{Fun}\, A\, F} f) & = & \lambda y.\ \mathsf{R}^{\mathsf{nf}}_{\xi}(\downarrow^{F \cdot a}(f \cdot a)) \\
\mathsf{R}^{\mathsf{nf}}_{(y,\xi)}(\downarrow^{\mathsf{Type}}(\mathsf{Fun}\, A\, F)) & = & (y : \downarrow^{\mathsf{Type}} A) \to \downarrow^{\mathsf{Type}}(F \cdot a)
\end{array}
$$

where $a = \uparrow^{A} y$.

Normalization for Dependent Types

- Canonical environment:

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

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

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

Correctness of Normalization

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

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

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

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

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

Partial Equivalence Relations

- Relation $A = A' \in \mathsf{Type}$ shall mean that A, A' are extensionally equal type values.
- Relation $a = a' \in A$ shall mean that a, a' are extensionally equal values of type A.
- Defined simultaneously by induction-recursion:

$$\frac{A = A' \in \mathsf{Type} \qquad F \cdot a = F' \cdot a' \in \mathsf{Type} \text{ for all } a = a' \in A}{\mathsf{Fun}\, A\, F = \mathsf{Fun}\, A'\, F' \in \mathsf{Type}}$$

$$\frac{f \cdot a = f' \cdot a' \in F \cdot a \text{ for all } a = a' \in A}{f = f' \in \mathsf{Fun}\, A\, F}$$

- Models $\beta\eta$-equality.

Typed Candidate Spaces

- Greatest and least PERs:

$$
\begin{array}{lcl}
d = d' \in \top & \iff & \mathsf{R}^{\mathsf{nf}}_{\xi}(d) =_\alpha \mathsf{R}^{\mathsf{nf}}_{\xi}(d') \text{ for all } \xi \\
e = e' \in \bot & \iff & \mathsf{R}^{\mathsf{ne}}_{\xi}(e) =_\alpha \mathsf{R}^{\mathsf{ne}}_{\xi}(e') \text{ for all } \xi
\end{array}
$$

- Greatest and least type candidate:

$$
\begin{array}{lcl}
a = a' \in \overline{A} & \iff & \downarrow^A a = \downarrow^A a' \in \top \\
a = a' \in \underline{A} & \iff & a = \uparrow^A e \text{ and } a' = \uparrow^A e' \text{ and } e = e' \in \bot
\end{array}
$$

- Sandwich property:

$$
a = a' \in \underline{A} \implies a = a' \in A \implies a = a' \in \overline{A}
$$

Completeness of Normalization

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

$$
\begin{array}{rcl}
\Gamma \vdash t = t' : T & \Longrightarrow & \overbrace{(\!|t|\!)_{\rho_\Gamma}}^{a} = \overbrace{(\!|t'|\!)_{\rho_\Gamma}}^{a'} \in \overbrace{(\!|T|\!)_{\rho_\Gamma}}^{A} \\
& \Longrightarrow & a = a' \in \overline{A} \\
& \Longrightarrow & \downarrow^A a = \downarrow^A a' \in \top \\
& \Longrightarrow & \mathsf{R}^{\mathsf{nf}}_{\xi_\Gamma} \downarrow^A a =_\alpha \mathsf{R}^{\mathsf{nf}}_{\xi_\Gamma} \downarrow^A a'
\end{array}
$$

Soundness of Normalization

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

$$
\begin{array}{rcl}
\Gamma \vdash t : T & \Longrightarrow & \Gamma \vdash t : T \circledR \overbrace{(\!| t |\!)_{\rho_\Gamma}}^{a} \in \overbrace{(\!| T |\!)_{\rho_\Gamma}}^{A} \\
& \Longrightarrow & \Gamma \vdash t = \mathsf{R}^{\mathsf{nf}}_{\xi_\Gamma} {\downarrow^A} a : T \\
& \Longleftrightarrow & \Gamma \vdash t = \mathsf{nf}^T_\Gamma(t) : T
\end{array}
$$

Impredicative Polymorphism

- Impredicativity: Quantification over all types gives a type.

$$\frac{\Gamma, X : \mathsf{Type} \vdash T : \mathsf{Type}}{\Gamma \vdash (\forall X : \mathsf{Type}.\ T) : \mathsf{Type}}$$

- Applications:
  - (Functional) programming: System F, Haskell, ...
  - Second-order logic.
- Semantic difficulty: Valid types cannot be defined from below.

$$\frac{F \cdot A = F' \cdot A' \in \mathsf{Type} \text{ for all } A = A' \in \mathsf{Type}}{\forall F = \forall F' \in \mathsf{Type}}$$

- Circularity!

NbE for System F

- Semantic type candidate $\mathcal{A}$ for S

$$\underline{S} \subseteq \mathcal{A} \subseteq \overline{S}$$

- Interpret $\forall$ by quantifying over all candidates (Girard):

$$[\![\forall X T]\!]\rho = \bigcap_{\underline{S} \subseteq \mathcal{A} \subseteq \overline{S}} [\![T]\!](\rho, \mathcal{A}/X)$$

Results

- NbE for dependent types and impredicativity: $\mathsf{CoC} + \mathbb{N}$.
  (Close to Coq's logical basis).
- Decidability of type checking with η on type level.
- Singleton types and universes.
- Theoretical basis for "compiled reduction" with η.

Future Work

- Agda: compiled equality checking based on NbE.
- Full Calculus of Inductive Constructions (Coq).
- Use NbE-semantics as tool to develop sound extensions of dependent type systems.

Acknowledgements

- To the mentorate: Martin Hofmann, Helmut Schwichtenberg, Peter Dybjer.
- To my coauthors: Klaus Aehlig, Peter Dybjer, Thierry Coquand, Michael Pagano.
- To my boss: Martin Hofmann.
- To my colleagues, family and friends.

A Munich Topic

- Helmut Schwichtenberg, Ulrich Berger
- Thorsten Altenkirch, Martin Hofmann, Thomas Streicher
- Mattias Eberl (PhD)
- Klaus Aehlig, Felix Joachimski
- Freiric Barral (PhD)
- Florian Haftmann, Tobias Nipkow