Normalization by Evaluation and the Foundations of Constructive Mathematics 1972 - 2009

Peter Dybjer

Chalmers tekniska högskola, Göteborg, Sweden

Workshop on Normalization by Evaluation
August 15, 2009
University of California at Los Angeles

The question

Is normalization by evaluation an intrinsic part of a foundational framework for constructive mathematics?

The question

Is normalization by evaluation an intrinsic part of a foundational framework for constructive mathematics?

- in 1972 - 73

The question

Is normalization by evaluation an intrinsic part of a foundational framework for constructive mathematics?

- in 1972 - 73
- progress 1979 - 2009

The question

Is normalization by evaluation an intrinsic part of a foundational framework for constructive mathematics?

- in 1972 - 73
- progress 1979 - 2009
- an open problem

It's 1972. State of the art

Normalization proofs with "computablility" predicates:

It's 1972. State of the art

Normalization proofs with "computablility" predicates:

- Tait 1967 - simply typed lambda calculus
- Girard 1971 - system F

It's 1972. State of the art

Normalization proofs with "computablility" predicates:

- Tait 1967 - simply typed lambda calculus
- Girard 1971 - system F

Formulas as types:

It's 1972. State of the art

Normalization proofs with "computablility" predicates:

- Tait 1967 - simply typed lambda calculus
- Girard 1971 - system F

Formulas as types:

- Howard 1968, intuitionistic predicate logic with equality
- de Bruijn 1968, Automath
- Scott 1970, Constructive Validity

It's 1972. State of the art

Normalization proofs with "computablility" predicates:

- Tait 1967 - simply typed lambda calculus
- Girard 1971 - system F

Formulas as types:

- Howard 1968, intuitionistic predicate logic with equality
- de Bruijn 1968, Automath
- Scott 1970, Constructive Validity

Intuitionistic type theory with normalization proofs (unpublished):

It's 1972. State of the art

Normalization proofs with "computablility" predicates:

- Tait 1967 - simply typed lambda calculus
- Girard 1971 - system F

Formulas as types:

- Howard 1968, intuitionistic predicate logic with equality
- de Bruijn 1968, Automath
- Scott 1970, Constructive Validity

Intuitionistic type theory with normalization proofs (unpublished):

- Martin-Löf 1971, A theory of types. Dependent type theory with type : type. Girard's paradox.
- Martin-Löf 1972, An intuitionistic theory of types. Predicative theory with a universe of small types. Published in 1997!

About models for intuitionistic type theories ... (p82)

In the study of models of intuitionistic theories, one has the choice between classical and intuitionistic abstractions on the metalevel. ... An obstacle to the formulation of a general intuitionistic notion of model has been the lack of a sufficiently welldeveloped intuitionistic notion of set.

Using the type-theoretic abstractions described in [17], I intend in the following to formulate an intuitionistic notion of model.

... and the notion of definitional equality (p82-83)

The transistion to intuitionistic abstractions on the metalevel is both essential and nontrivial. Essential, because in what seems to me to be the most fruitful notion of model, the interpretation of the convertibility relation conv *is standard, that is, it is interpreted as definitional equality* $=_{\mathrm{def}}$ *in the model, and definitional equality is a notion which is unmentionable within the classical set theoretic framework.*

Term model of the positive implicational calculus (p 87)

(a) $\mathrm{Typ} =_{\mathrm{def}}$ the type of pairs (A, ϕ), where A is a type symbol and ϕ a species of closed terms with type symbol A.

(b) $\mathrm{Obj}((A, \phi)) =_{\mathrm{def}} (\Sigma a \in \mathrm{Term}(A))\phi(a)$.

(c) $F((A, \phi), (B, \psi)) =_{\mathrm{def}} (A \to B$, the species of all closed terms b with type symbol $A \to B$ such that

$$(\forall x \in \mathrm{Obj}((A, \phi)))(\exists y \in \mathrm{Obj}((B, \psi)))(b(p(x))\,\mathrm{red}\,p(y))$$

(d) $\mathrm{Ap}(b, a) =_{\mathrm{def}} p(q(b, a))$

(e) ... $\overline{K} =_{\mathrm{def}} (K, (\lambda x)((K(p(x)), (\lambda y)(x,$
the proof that $K(p(x), p(y))\,\mathrm{red}\,p(x))$,
the proof that $K(p(x))\,\mathrm{red}\,K(p(x)))))$

You have just witnessed the birth of nbe!

... in the term model, we achieve that if a conv *b, then the normal forms of a and b as well as the proofs which show that they are computable (hereditarily normalizable) are definitionally equal.*

Local formalizability, p 99

The proof of normalization for my intuitionistic type theory (see [17]) becomes locally formalizable in itself. When the dubious rule of lambda conversion was allowed, I could not carry out the proof of normalization for every specific term in the theory itself, contrary to what one would expect from one's experience with other full scale formal theories. I was only able to prove C_A and C_B to be extensionally equal, whereas one would like to have $C_A =_{\mathrm{def}} C_B$. Here C_A and C_B are the computability predicates associated with the type symbols A and B, respectively.

An Intuitionistic Theory of Types: Predicative Part

Logic Colloquium 1973 in Bristol.

An Intuitionistic Theory of Types: Predicative Part

Logic Colloquium 1973 in Bristol.

- First published version of Martin-Löf's intuitionistic type theory. (Super) combinator version, no bound variables.

An Intuitionistic Theory of Types: Predicative Part

Logic Colloquium 1973 in Bristol.

- First published version of Martin-Löf's intuitionistic type theory. (Super) combinator version, no bound variables.
- Normalization "nbe-style". Not only proved that normal forms exist, but they were explicitly given ("computed").

An Intuitionistic Theory of Types: Predicative Part

Logic Colloquium 1973 in Bristol.

- First published version of Martin-Löf's intuitionistic type theory. (Super) combinator version, no bound variables.
- Normalization "nbe-style". Not only proved that normal forms exist, but they were explicitly given ("computed").
- Several meta-theoretic results proved as corollaries of nbe
  - 3.7 Church-Rosser (Hancock)
  - 3.8 Decidability of convertibility
  - 3.13 Decidability of the $\in$-relation (type-checking algorithm)

The model of closed normal terms, the normalization theorem (for closed terms) and its consequences

In the present theory, however, the definiton of the notion of convertibility and the proof that an arbitrary term is convertible can no longer be separated, because the type symbols and the terms are generated simultaneously. Instead we shall show by induction on the length of a closed derivation, if it ends with $a \in A$, how to define a' and a'', where

- *a' is a closed normal term with type symbol A', called the normal form of a, such that a red a', and*
- *a'' is a proof of $A''(a')$, which it is sometimes more natural to think of as an object of type $A''(a')$,*

and if it ends with $a \operatorname{conv} b$, that

$$a' =_{\mathrm{def}} b' \quad \text{and} \quad a'' =_{\mathrm{def}} b''$$

Why did not the story end in 1973?

- (Super) combinator version of intuitionistic type theory was considered too weak to be useful.
- Local formalizability of nbe-proof (and its corollaries) is claimed but not shown.
- No consistency proof relative to set theory.

1979 revolution - Constructive Mathematics and Computer Programming

- Meaning explanations! Lazy evaluation of *closed* terms to *constructor* form.
- Extensional type theory - a stronger theory
- Normalization of open expressions not part of meaning theory. Normalization and decidability of judgement do not hold.
- Nbe forgotten (implementations like NuPRL and GTT did not employ normalization).
- Against metatheory!

1984 impredicative uprising - normalization is back!

- Coquand and Huet 1984. A calculus of constructions. Intuitionistic type theory: impredicative part.
- Normalization property and decidable judgements, used for type-checking in the implementation.
- No discussion of intuitionistic model theory and nbe. Set-theoretic abstractions on the meta level.

1986 counter revolution

- Martin-Löf 1986. Intuitionistic type theory based on a (predicative) logical framework.
- Intensional type theory. Decidable judgements considered philosophically important. Normalization part of type-checking in the ALF implementation of 1990.
- Combined legacy of 1973 (decidability) and 1979 (meaning explanations) but metatheory not written down by Martin-Löf. No revised version of 1973 paper.

NBE (re)discovered, programs written

- 1991 Berger and Schwichtenberg. Nbe for betaeta. Scheme program and set-theoretic proof.
- 1993 Coquand and Dybjer. 1972-style nbe in dependently typed metalanguage.
  - Extracted an nbe-algorithm for positive implicational calculus (typed combinatory logic).
  - Added data types, including Brouwer ordinals
  - Implemented algorithm + correctness proof in ALF. Both integrated and external version.

Also tried to formalize 1973 version of intuitionistic type theory, but could not do it for dependent types, only for the simply typed part. Tried to do the "local formalizability" but got stuck.

Barras, Coq in Coq, MSc 1994

Local formalizability of Calculus of Constructions!

> *In particular, we prove strong normalization and decidability of type inference. From the latter proof, we extract a certified Caml Light program, which performs type inference (or type-checking) for an arbitrary typing judgement in CC. Integrating this program in a larger system, including a parser and pretty-printer, we obtain a stand-alone proof-checker, called CoC, for the Calculus of Constructions. As an example, the formal proof of Newman's lemma, build with Coq, can be re-verified by CoC with reasonable performance.*

Definition of the theory as a reduction system. CoC not ITT.

Local formalizability - requirements

Local formalizability of the main meta-theorems (in particular decidability of type-checking) for a fragment Martin-Löf type theory in a suitable larger fragment.

Canonical object and meta theory.

Machine-checked proof.

Confluence of theory and practice. The type-checking algorithm should be (close to) one used in practice.

Make good use of the meta language. Make good use of dependent types. Mathematically elegant treatment, category theory?

NBE for dependent types with beta and eta

- 2004 Martin-Löf. Nbe for intuitionistic type theory with betaeta BS91 + ML73. Paper proof.
- 2006 Danielsson. Nbe for intuitionistic type theory with betaeta in Agda using non-strictly positive types.
- 2007 Abel, Aehlig, Dybjer, 2007 Abel, Coquand, Dybjer. Haskell algorithm, set theoretic proof.

Relevant progress on several different topics

We have looked at

- Improved object theory
- Type-checking
- Nbe

But the following is also relevant:

- Induction-recursion (for metatheory)
- Categorical models of dependent type theory (leading up to cwfs)
- Intuitionistic model theory (setoids and other E-notions, E-categories, E-cwfs)

Categorical model for type-checking dependent types

- 2008 Abel, Coquand, Dybjer. Cwf-style type-checking algorithm using hereditary substitution. Specification of correctness as initiality of cwfs.

We shall here show only the core of dependent type theory, sometimes called "the logical framework" (LF). The methods generalize to deal with the "data types" of Martin-Löf type theory and extensions with inductive definitions. We use Russell-style terms (type symbols are just special terms) defined in Haskell

```
data Tm = Var Int | App Tm Tm
        | Lam Tm | Pi Tm Tm | U
```

Normal and neutral raw terms

Normal terms t (including normal types) have no β-redexes. They are inductively generated together with the auxiliary subclass of neutral terms s:

$$\begin{array}{lcl} t & ::= & s \mid \lambda(t) \mid \Pi(t,t) \mid \mathrm{Set} \\ s & ::= & i \mid \mathrm{ap}(s,t) \end{array}$$

In Haskell:

```
data No = Ne Ne | Lam No | Pi No No | U
data Ne = Var Int | App Ne No
```

Note again *Russell-style*.

The type-checking functions

For simplicity, we use a single type `Tm` containing both ordinary terms and type symbols, both normal and non-normal ones. But the algorithm is only intended to be applied to normal ones.

```
type Ty    = Tm
type Cxt   = [Ty]
type Subst = [Tm]

isCo :: Cxt -> Bool
isSu :: Cxt -> Cxt -> Subst -> Bool
isTy :: Cxt -> Ty -> Bool
isTm :: Cxt -> Ty -> Tm -> Bool
```

Checking contexts

```
isCo :: Cxt -> Bool
isCo []      = True
isCo (a:cxt) = isCo cxt && isTy cxt a
```

checks whether a list of expressions represents a correct context.

Checking types

```
isTy :: Cxt -> Ty -> Bool
isTy cxt (Pi a b) = isTy cxt a && isTy (a:cxt) b
isTy cxt U        = True
isTy cxt a        = isTm cxt U a
```

checks whether an expression is a correct type with respect to a context.

Checking terms

```
isTm :: Cxt -> Ty -> Tm -> Bool
isTm cxt (Pi a b) (Lam t)   = isTm (a:cxt) b t
isTm cxt _        (Lam t)   = False
isTm cxt U        (Pi a b)  = isTm cxt U a &&
                              isTm (a:cxt) U b
isTm cxt _        (Pi a b)  = False
isTm cxt _        U         = False
isTm cxt a        s         = case inferTy cxt s of
                                Just a' -> a == a'
                                Nothing -> False
```

checks whether an expression has a type with respect to a context.

Infering the type of a neutral term

The type-checking algorithm is as usual bi-directional: to check whether a neutral term has a given type we try to infer the type of the function and then check whether it matches the type of the argument.

```
inferTy :: Cxt -> Tm -> Maybe Ty
inferTy cxt (Var i)   = Just (shift (cxt !! i) (i+1))
inferTy cxt (App s t) = case inferTy cxt s of
   Just (Pi a b) -> if isTm cxt a t
                    then Just (hsubst b (t:ide))
                    else Nothing
   otherwise     -> Nothing
```

It calls three auxiliary function

```
hsubst :: Ty -> Subst -> Ty
ide    :: Subst
shift :: Ty -> Int -> Ty
```

where `shift a i` increases all free variables in `a` by `i`.

Hereditary substitution

Substitution which preserves *normality*:

```
hsubst :: Tm -> Subst -> Tm
hsubst (Var i)   ts = ts !! i
hsubst (App s t) ts = app (hsubst s ts) (hsubst t ts)
hsubst (Lam t)   ts = Lam (hsubst t (lift ts))
hsubst (Pi a b)  ts = Pi (hsubst a ts) (hsubst b (lift ts))
hsubst U         ts = U
```

which uses more auxiliary functions `lift` and

```
app  :: Tm -> Tm -> Tm
app (Lam t) s = hsubst t (s:ide)
app r       s = App r s
```

Abel, Coquand, Pagano 2009: type checking using nbe

Instead of hereditary substitution of Abel, Coquand, Dybjer 2008

```
hsubst b (t:ide)
```

Abel, Coquand, Pagano 2009 use explicit substitution and nbe

```
nbeTy (ESubst b (t:ide))
```

Correctness of the type-checking algorithm is then proved as a corollary of the correctness of nbe.

Nbe for intuitionistic type theory

First order terms (subset of normal terms is used)

```
data Tm = Var Int | App Tm Tm
        | Lam Tm | Pi Tm Tm | U
```

Semantic domain of values are terms in higher order syntax (subset of normal terms is used)

```
data D = VarD Int | AppD D D
       | LamD (D -> D) | PiD D (D -> D) | UD
```

Again, neutrals and normals are thrown together in one type here. We could separate them for pedagogical purpose.

Reification and eta-expansion

Abel, Aehlig, Dybjer 2007:

```
TM = Int -> Tm

reifyTy :: D -> TM
reify   :: D -> D -> TM
reflect :: D -> TM -> D
```

Abel, Coquand, Pagano 2009 separate eta-expansion and reification:

```
downTy :: D -> D
down   :: D -> D -> D
up     :: D -> D -> D

readback :: D -> TM
```

Eta-expansion in the model

```
downTy (PiD a f) = PiD (downTy a) (\d -> downTy (f (up a d))
downTy UD        = UD
downTy d         = d

down (PiD a f) d = LamD (\e -> down (f (up a e))
                                    (appD d (up a e)))
down UD        d = downTy d
down e         d = d

up (PiD a f) e = LamD (\d -> up (f d) (AppD e (down a d)))
up d         e = e
```

Readback

```
readback :: D -> TM
readback (VarD i)   n = Var (n - i - 1)
readback (AppD d e) n = App (readback d n) (readback e n)
readback (LamD f)   n = Lam (readback (f (VarD n)) (n + 1))
readback (PiD a f)  n = Pi (readback a n)
                           (readback (f (VarD n)) (n + 1))
readback UD         n = U
```

`n` is the maximal number of free variables, `i` is the de Bruijn *level*.

Explicit substitution calculus (for cwfs)

We can also define implicit syntax. Let's do it in Haskell:

```
data Tm = ESubst Tm Ts | Q | App Tm Tm
        | Lam Tm | Pi Tm Tm | U

data Ts = Empty | Ext Ts Tm | P | Comp Ts Ts | Id
```

We will define the normalization function for this.

Evaluation in cwf of normal forms in hoas

```
eval :: Tm -> [D] -> D

eval (ESub a as) ds = eval a (evals as ds)
eval Q           ds = head ds
eval (Pi a b)    ds = PiD (eval a ds) (\d -> eval b (d:ds))
eval (App r s)   ds = appD (eval r ds) (eval s ds)
eval (Lam r)     ds = LamD (\d -> eval r (d:ds))
eval U           ds = UD

evals :: Ts -> [D] -> [D]
...

nbeTy :: Ctx -> Ty -> Ty
nbeTy ctx a = readback (downTy (eval a (idenv n))) n
    where n = length ctx
```

Inductive-recursive definition of equal well-formed types and equal well-typed terms

We define by a simultaneous inductive-recursive definition partial equivalence relations

- $\approx \subseteq D \times D$ (equal total types)
- $\approx_{a,a'} \subseteq D \times D$ (equal total terms of equal total types $a \approx a'$).

Yields completeness proof for nbe. For soundness another inductive-recursively defined relation between Tm and D is needed.
In our papers on nbe for dependent types the meta language is always set theory. The challenge is to do the analogous thing in type theory.

Nbe for dependent type theory with Agda as metalanguage

PhD thesis of Danielsson 2007. Nbe with correctness proof.

Object language Very explicit syntax following Curien 1993 "Substitution up to Isomorphism", E-cwf of Dybjer 1996 "Internal Type Theory" :

$$(::^{=}_{\vdash}) : (\Gamma : Cxt) \to (\tau_1, \tau_2 : Ty\ \Gamma) \to \Gamma \vdash \tau_1 \to \tau_1 =_* \tau_2 \to \Gamma \vdash \tau_2$$

Meta language AgdaLight allowing inductive definition of syntactic domain above and semantic domain which is not a strictly positive inductive defintion and shouldn't be accepted by disciplined meta language (cf Haskell type D). Informal argument why this is ok, and why it should be possible to eliminate.

A categorical formulation of the type-checking theorem for intuitionistic type theory

Conjecture: The type-checking cwf (with Π and U-types) is initial among E-cwfs can be proved if we add suitable indexed inductive-recursive definitions to the cwf.

object theory Initial E-cwf with $U, \Pi, \ldots$, see "Internal Type Theory" (Dybjer 1996)

meta theory Intuitionistic type theory with indexed inductive-recursive definitions (Dybjer 2000, Dybjer and Setzer 2001,2006)

Back to the original question

Is normalization by evaluation an intrinsic part of a foundational framework for constructive mathematics?

Back to the original question

Is normalization by evaluation an intrinsic part of a foundational framework for constructive mathematics? It remains to be seen!

Back to the original question

Is normalization by evaluation an intrinsic part of a foundational framework for constructive mathematics? It remains to be seen!

- Nbe was an intrinsic part of the foundational framework presented by Martin-Löf in 1973. nbe is the prefered way to express normalization in a type-theoretic setting

Back to the original question

Is normalization by evaluation an intrinsic part of a foundational framework for constructive mathematics? It remains to be seen!

- Nbe was an intrinsic part of the foundational framework presented by Martin-Löf in 1973. nbe is the prefered way to express normalization in a type-theoretic setting
- It ought to be put back. But it has not yet been done.

Back to the original question

Is normalization by evaluation an intrinsic part of a foundational framework for constructive mathematics? It remains to be seen!

- Nbe was an intrinsic part of the foundational framework presented by Martin-Löf in 1973. nbe is the prefered way to express normalization in a type-theoretic setting
- It ought to be put back. But it has not yet been done.
- less clear if we have other style of foundational framework (Feferman-Aczel style)

Final thoughts on metamathematical modelling intuitionistically

What is going on in some primitive "pre-mathematical " sense?
The difference between a language, and a language embedded in another language. Static vs dynamic aspects of a language.
Intuitionistic logic has real meaning not just an ephemeral one.
Do we need a formal metatheory? Well, it's a full-scale framework, so should be able to account for everything "locally".