Normalization by Evaluation
for Martin-Löf Type Theory
with One Universe

Peter Dybjer, Göteborg
(with Andreas Abel, Munich, and Klaus Aehlig, Swansea)

MFPS XXIII, New Orleans
11-14 April, 2007
Let us define $\text{power} \ m \ n = m^n$.

\[
\text{power} :: \text{int} \rightarrow \text{int} \rightarrow \text{int}
\]

\[
\text{power} \ m \ 0 = 1 \\
\text{power} \ m \ (\text{Succ} \ n) = m \times (\text{power} \ m \ n)
\]

In Gödel System T

\[
\text{power} \ m \ n = \text{rec} \ 1 \ (\lambda x \ y \rightarrow m \times y) \ n
\]

Let $n = 3$. Simplify:

\[
\text{power} \ m \ 3 = m \times (m \times m)
\]

by using the reduction rules for $\text{power}$, $\times$, and $\text{+}$. 
In Martin-Löf type theory we can define the type-valued function $\text{Power } A \ n = A^n$. *Set* is the type of small types - a universe:

\[
\text{Power} :: \text{Set} \to \text{Nat} \to \text{Set}
\]

\[
\text{Power } A \ 0 = 1 \\
\text{Power } A \ (\text{Succ } n) = A \times (\text{Power } A \ n)
\]

\[
\text{Power } A \ n = \text{rec } 1 (\lambda x \ y \to A \times y) \ n
\]

Let $n = 3$. Simplify:

\[
\text{Power } A \ 3 = A \times (A \times (A \times 1))
\]

by using the reduction rules for \text{Power}. Can we simplify further?
Normalization during type-checking

To check that

\[(2007, (4, (12, ()))) :: \text{Power Nat 3}\]

we need to normalize the type:

\[(2007, (4, (12, ()))) :: \text{Nat } \ast (\text{Nat } \ast (\text{Nat } \ast 1))\]
Programming normalization – by evaluation

Normalization as a *program*! Constructive metamathematics is meta-programming!

An elegant way is to normalize by “evaluating” a term in a model, and then extracting the normal form:

\[
\begin{array}{c}
\text{syntax} \xrightarrow{\mathcal{L}} \text{model} \\
\downarrow \\
nbe t = \downarrow \llbracket t \rrbracket
\end{array}
\]

In this talk we shall view the model as the *model of normal forms in higher-order abstract syntax*. 
Plan


- Normalization algorithms for terms and types:
  - $\text{nbe}_\Gamma^A t = \downarrow_\Gamma [A]_\rho [t]_\rho$
  - $\text{Nbe}_\Gamma A = \downarrow_\Gamma [A]_\rho$

- Correctness of normalization algorithm for terms and types means decidability of equality:
  - If $\Gamma \vdash t, t' : A$ then $t \equiv_{\beta\eta} t'$ iff $\text{nbe}_\Gamma^A t \equiv \text{nbe}_\Gamma^A t' \in Tm$.
  - If $\Gamma \vdash A, A'$ then $A \equiv_{\beta\eta} A'$ iff $\text{Nbe}_\Gamma A \equiv \text{Nbe}_\Gamma A' \in Tm$. 
Types and terms with de Bruijn indices (types are terms - universe à la Russell)

\[
Tm \ni r, s, t, z, A, B \quad ::= \quad \nu_i \quad \text{de Bruijn index}
\mid \lambda t \quad \text{abstracting 0th variable}
\mid r s \quad \text{application}
\mid Zero \quad \text{natural number “0”}
\mid Succ t \quad \text{successor}
\mid Rec A z s t \quad \text{primitive recursion}
\mid \Pi AB \quad \text{dependent function type}
\mid Nat \quad \text{natural number type}
\mid Set \quad \text{universe}
\]

We can add other set constructors too: \(\Sigma AB, A + B, 0, 1\), and inductively defined datatypes. (E.g example with \(Power\)-types used \(\times\).)
Reduction and conversion

One-step $\beta\eta$-reduction $t \longrightarrow t'$ is given as the congruence-closure of the following contractions.

- $(\lambda t)s \longrightarrow t[s]$ (β-λ)
- $\lambda. (\uparrow^1 t) \nu_0 \longrightarrow t$ (η)
- $Rec\ A\ z\ s\ Zero \longrightarrow z$ (β-Rec-Zero)
- $Rec\ A\ z\ s\ (Succ\ r) \longrightarrow s\ r\ (Rec\ A\ z\ s\ r)$ (β-Rec-Succ)

Its reflexive-transitive closure $\longrightarrow^*$ is confluent, so we can define $t =_{\beta\eta} t'$ as $\exists s. t \longrightarrow^* s^* \leftarrow t'$. 
Judgement forms

\[\Gamma \vdash \Gamma \text{ is a well-formed context}\]
\[\Gamma \vdash A \quad A \text{ is a well-formed type in context } \Gamma\]
\[\Gamma \vdash t : A \quad t \text{ has type } A \text{ in context } \Gamma\]

We follow Martin-Löf 1972: basis is *conversion of untyped terms* (does not count as judgement):

\[t =_{\beta\eta} t'\]

Martin-Löf 1973 and onwards instead has *typed equality judgements*

\[\Gamma \vdash A = A'\]
\[\Gamma \vdash t = t' : A\]
Some inference rules

We only give the rules for well-formed sets

\[ \Gamma \vdash \text{Nat} : \text{Set} \]
\[ \Gamma, A \vdash B : \text{Set} \]
\[ \Gamma \vdash \Pi AB : \text{Set} \]

well-formed types

\[ \Gamma \vdash A : \text{Set} \]
\[ \Gamma \vdash A : \text{Set} \]
\[ \Gamma \vdash \Pi AB \]

and the type conversion rule:

\[ \Gamma \vdash t : A, \Gamma \vdash A' \]
\[ \Gamma \vdash t : A' \]
\[ A = \beta \eta A' \]

There are also introduction and elimination rules for \( \Pi \) and \( \text{Nat} \), and rules for context formation and assumption.
Semantics: normal forms in higher order abstract syntax

First-order syntax of normal and neutral (well-formed) types and (well-typed) terms:

\[
A, B, t, u ::= \Pi A B \mid \text{Nat} \mid \text{Set} \mid \lambda t \mid \text{Zero} \mid \text{Succ} t \mid s
\]

\[
s ::= v_i \mid s t \mid \text{Rec} A t u s
\]

"There is no model of normal forms; normality is not closed under application (and recursion)."

Define a domain \( D \) of normal forms in higher-order abstract syntax with the following "constructors":

\[
\begin{align*}
\text{Pi} & : D \times [D \to D] \to D \\
\text{Nat} & : D \\
\text{Set} & : D \\
\text{Lam} & : [D \to D] \to D \\
\text{Zero} & : D \\
\text{Succ} & : D \to D \\
\text{Ne} & : TM_\bot \to D
\end{align*}
\]

where \( TM = N \to Tm_\mathbb{Z} \) (See paper for strictness issues.)
Haskell datatypes for terms and normal forms in hoas

data Tm = Var Int | App Tm Tm | Lam Tm |
        | Zero   | Succ Tm | Rec Tm Tm Tm Tm |
        | Nat    | Pi Tm Tm | Set |
        deriving (Show, Eq)

type TM = Int -> Tm

data D = PiD D (D -> D) -- dependent function type |
        | NatD          -- natural number type |
        | SetD          -- type of sets |
        | LamD (D -> D) -- function |
        | ZeroD         -- 0 |
        | SuccD D       -- successor |
        | NeD TM        -- neutral terms
Nbe functions in Haskell

A context is a list of types

\[
\text{type Cxt} = [\text{Tm}]
\]

Normalization of a term wrt a type and a context:

\[
\text{nbe} :: \text{Cxt} \rightarrow \text{Tm} \rightarrow \text{Tm} \rightarrow \text{Tm}
\]

Normalization of a type wrt a context

\[
\text{nbeT} :: \text{Cxt} \rightarrow \text{Tm} \rightarrow \text{Tm}
\]
Evaluation function

\[
\begin{align*}
\llbracket - \rrbracket_\rho &: \text{Tm} \rightarrow \llbracket \text{N} \rightarrow \text{D} \rrbracket \rightarrow \text{D} \\
\llbracket v_i \rrbracket_\rho &= \rho(i) \\
\llbracket \lambda t \rrbracket_\rho &= \text{Lam} (d \mapsto \llbracket t \rrbracket_\rho, d) \\
\llbracket r \cdot s \rrbracket_\rho &= \llbracket r \rrbracket_\rho \cdot \llbracket s \rrbracket_\rho \\
\llbracket \text{Zero} \rrbracket_\rho &= \text{Zero} \\
\llbracket \text{Succ} t \rrbracket_\rho &= \text{Succ} \llbracket t \rrbracket_\rho \\
\llbracket \text{Rec A z s t} \rrbracket_\rho &= \text{rec} (d \mapsto \llbracket A \rrbracket_\rho, d) \llbracket Z \rrbracket_\rho \llbracket s \rrbracket_\rho \llbracket t \rrbracket_\rho \\
\llbracket \Pi A B \rrbracket_\rho &= \text{Pi} \llbracket A \rrbracket_\rho (d \mapsto \llbracket B \rrbracket_\rho, d) \\
\llbracket \text{Nat} \rrbracket_\rho &= \text{Nat} \\
\llbracket \text{Set} \rrbracket_\rho &= \text{Set}
\end{align*}
\]
Application of normal forms in hoas

We define application on D as the function

\[
\text{app} : [D \rightarrow [D \rightarrow D]]
\]

\[
(Lam f) \cdot d = f d
\]

\[
e \cdot d = \perp \text{ if } e \text{ is not Lam } f
\]

where in the following “default \( \perp \) clauses” like the last one are always tacitly assumed.

In Haskell:

\[
\text{appD} :: D \rightarrow D \rightarrow D
\]

\[
\text{appD} (\text{LamD } f) \ d = f \ d
\]

We also need to define primitive recursion rec in the model, but first we need reification and reflection.
Reification - translating hoas to foas

\[ \downarrow : [D \rightarrow TM_{\perp}] \]

\[ \downarrow_k (\Pi a g) = \Pi(\downarrow_k a)(\downarrow_{k+1} g(\uparrow^a \hat{\nu}_{-(k+1)})) \]

\[ \downarrow_k \text{Nat} = \text{Nat} \]

\[ \downarrow_k \text{Set} = \text{Set} \]

\[ \downarrow_k (\text{Ne} \hat{t}) = \hat{t}(k) \]

\[ \downarrow : [D \rightarrow [D \rightarrow TM_{\perp}]] \]

\[ \downarrow_k^\text{Set} a = \downarrow_k a \]

\[ \downarrow_k^{\Pi a g} (\text{Lam} f) = \lambda(\downarrow_{k+1} g(\uparrow^a \hat{\nu}_{-(k+1)}))(f(\uparrow^a \hat{\nu}_{-(k+1)})) \]

\[ \downarrow_k^{\text{Nat} \text{Zero}} = \text{Zero} \]

\[ \downarrow_k^{\text{Nat} (\text{Succ} d)} = \text{Succ}(\downarrow_k^{\text{Nat} d}) \]

\[ \downarrow_k^c (\text{Ne} \hat{t}) = \hat{t}(k) \]

if \( c \neq \perp, c \neq \Pi \ldots \)
Mapping neutral terms (including variables) to D:

\[ \uparrow : [D \rightarrow [TM \_ \_ \_ \rightarrow D]] \]

\[ \uparrow^{\text{Pi} a \ g} \hat{t} = \text{Lam}(d \mapsto \uparrow^{g(d)}(\hat{t} \_ \_ \_ \_ a \ d)) \]

\[ \uparrow^{c} \hat{t} = \text{Ne} \hat{t} \quad \text{if } c \neq \bot, c \neq \text{Pi} \ldots \]

We perform \( \eta \)-expansion. Hence we need the first argument which is a normal type in hoas - an element of D.
Primitive recursion on normal forms in hoas

\[
\text{rec} : \left[[D \rightarrow D] \rightarrow [D \rightarrow [D \rightarrow D]]\right]
\]

\[
\text{rec } a d_z d_s \text{ Zero } = d_z
\]

\[
\text{rec } a d_z d_s (\text{Succ } e) = d_s \cdot e \cdot (\text{rec } a d_z d_s e)
\]

\[
\text{rec } a d_z d_s (\text{Ne } \hat{t}) = \uparrow^{a(\text{Ne } \hat{t})} (k \mapsto \text{Rec} \left( \downarrow^a_{k+1} a(\text{Ne } v_{(k+1)}) \right) \\
(\downarrow^a_k d_z) \\
(\Pi \text{Nat}(d \mapsto a d \Rightarrow a(\text{Succ } d)) \, d_s) \\
\hat{t}(k))
\]

Here we use reification $\downarrow$ and reflection $\uparrow$. 
Normalization by evaluation for terms and types is now implemented by these two functions:

\[
\begin{align*}
nbe^\Gamma_A t & := \downarrow_{|\Gamma|} [t]_{\rho^\Gamma} \\
Nbe^\Gamma A & := \downarrow_{|\Gamma|} [A]_{\rho^\Gamma}
\end{align*}
\]

where \(\rho^\Gamma\) is the identity valuation which is obtained by reflection of the identity substitution.
Correctness of normalization function

Correctness means decidability of equality (convertibility of types and terms).

- If $\Gamma \vdash t, t' : A$ then $t =_{\beta\eta} t'$ iff $\text{nbe}_A^\Gamma t \equiv \text{nbe}_A^\Gamma t' \in Tm$.
- If $\Gamma \vdash A, A'$ then $A =_{\beta\eta} A'$ iff $\text{Nbe}_\Gamma A \equiv \text{Nbe}_\Gamma A' \in Tm$.

We split it up into two parts

Completeness

- If $\Gamma \vdash t, t' : A$ and $t =_{\beta\eta} t'$, then $\text{nbe}_A^\Gamma t \equiv \text{nbe}_A^\Gamma t' \in Tm$.
- If $\Gamma \vdash A, A'$ and $A =_{\beta\eta} A'$, then $\text{Nbe}_\Gamma A \equiv \text{Nbe}_\Gamma A' \in Tm$.

Soundness

- If $\Gamma \vdash t : A$ then $t =_{\beta\eta} \text{nbe}_A^\Gamma t$.
- If $\Gamma \vdash A$ then $A =_{\beta\eta} \text{Nbe}_\Gamma A$.

We will only discuss the former. The latter is shown by defining a Kripke logical relation between terms and their normal forms in hoas.
We inductively define $\mathbb{N} \mathit{at} \in \mathit{Per}$ by the following rules:

\[
\begin{align*}
\text{Zero} &= \text{Zero} \in \mathbb{N} \mathit{at} \\
\text{Succ} \, d &= \text{Succ} \, d' \in \mathbb{N} \mathit{at} \\
\text{Ne} \, \hat{t} &= \text{Ne} \, \hat{t} \in \mathbb{N} \mathit{at}
\end{align*}
\]

If we have a PER $\mathcal{A}$ and a family of PERs $\mathcal{G}(d)$ indexed by $d$ in the domain of $\mathcal{A}$, then we can build a PER of functions:

\[
\Pi \mathcal{A} \mathcal{G} = \{ (e, e') \mid (e \cdot d, e' \cdot d') \in \mathcal{G}(d) \text{ for all } (d, d') \in \mathcal{A} \}.
\]
We simultaneously define the PER $\text{Set} \in \text{Rel}$ and the family of PERS $[a]$ for $a$ in the domain of $\text{Set}$ by the following rules.

\[
\begin{align*}
a = a' \in \text{Set} & \quad g(d) = g'(d') \in \text{Set} \quad \text{for all } d = d' \in [a] \\
\Pi a \ g &= \Pi a' \ g' \in \text{Set} \\
\text{Nat} &= \text{Nat} \in \text{Set} \\
\text{Ne} \hat{t} &= \text{Ne} \hat{t} \in \text{Set}
\end{align*}
\]

\[
\begin{align*}
[Pi \ a \ g] &= \Pi [a] (d \mapsto [g(d)]) \\
[Nat] &= \mathcal{N}at \\
[Ne \hat{t}] &= \mathcal{N}e.
\end{align*}
\]
Inductive-recursive definition as monotone inductive definition

We define the graph $T \subseteq \mathcal{P}(D \times \text{Per})$ of $\lfloor \_ \rfloor$ inductively by the following rules.

\[
\begin{align*}
(a, \mathcal{A}) & \in T \\
(g(d), G(d)) & \in T \text{ for all } d \in \mathcal{A}
\end{align*}
\]

\[
(Pi\, a\, g, \Pi\mathcal{A}\, G) \in T
\]

\[
\begin{align*}
(Nat, \mathcal{Nat}) & \in T \\
(\text{Ne}\, t, \mathcal{N}e) & \in T
\end{align*}
\]

This is a monotone inductive definition using Aczel’s rule sets (see Handbook of Mathematical Logic).
Inductive-recursive definition of the PER of all types

This is like the definition of *small* types with some extra clauses:

\[
\begin{align*}
c &= c' \in \text{Set} \\
c &= c' \in \text{Type} \\
\text{Set} &= \text{Set} \in \text{Type}
\end{align*}
\]

\[
\begin{align*}
a &= a' \in \text{Type} \\
g(d) &= g'(d') \in \text{Type} \text{ for all } d = d' \in [a] \\
\Pi a \ g &= \Pi a' \ g' \in \text{Type}
\end{align*}
\]

\[
\begin{align*}
[\Pi a \ g] &= \Pi [a] (d \mapsto [g(d)]) \\
[\text{Nat}] &= \mathcal{N}at \\
[\text{Ne} \hat{=} t] &= \mathcal{N}e. \\
[\text{Set}] &= \text{Set}
\end{align*}
\]
Reification and reflection preserve equality

1. If \( c = c' \in \text{Type} \) then \( \uparrow^c \hat{t} = \uparrow^{c'} \hat{t} \in [c] \).
2. If \( c = c' \in \text{Type} \) then \( \downarrow c \equiv \downarrow c' \in TM \).
3. If \( c = c' \in \text{Type} \) and \( e = e' \in [c] \) then \( \downarrow^c e \equiv \downarrow^{c'} e' \in TM \).
Convertible terms are semantically related

- If $\Gamma \vdash A, A'$ and $A \equiv_{\beta\eta} A'$ then $\Gamma \models A = A'$.
- If $\Gamma \vdash t, t' : A$ and $t \equiv_{\beta\eta} t'$ then $\Gamma \models t = t' : A$.

where

$\Gamma \models A = A' \iff \Gamma \models A$ and $\forall \rho = \rho' \in [\Gamma]. \llbracket A \rrbracket_\rho = \llbracket A' \rrbracket_{\rho'} \in \text{Type}$

$\Gamma \models t = t' : A \iff \Gamma \models A$ and $\forall \rho = \rho' \in [\Gamma]. \llbracket t \rrbracket_\rho = \llbracket t' \rrbracket_{\rho'} \in \llbracket [A] \rrbracket_\rho$
Completeness of NbE

1. If $\Gamma \vdash t, t' : A$ and $t \equiv_{\beta\eta} t'$ then $\text{nbe}_\Gamma^A t \equiv \text{nbe}_\Gamma^A t' \in Tm$.

2. If $\Gamma \vdash A, A'$ and $A \equiv_{\beta\eta} A'$ then $\text{Nbe}_\Gamma A \equiv \text{Nbe}_\Gamma A' \in Tm$.

It follows that NbE is terminating on well-typed terms.
Conclusion

Key point. With nbe we get better tool for metatheory of type theory. It is more practical and more elegant.

- Extend Berger-Schwichtenberg style nbe to dependent types: normalize types as well as terms. Show that we can get eta for universe a la Russell. Key point for justifying Agda system.


- Key obstacle was overcome by starting with untyped nbe. (Note also that the algorithm for MLTT with only beta-conversion is more straightforward.)