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

Partial evaluation of programs

Let us define `power` $m\ n = m^n$.

```
power :: int -> int -> int

power m 0        = 1
power m (Succ n) = m * (power m n)
```

In Gödel System T

```
power m n = rec 1 (\x y -> m * y) n
```

Let $n = 3$. Simplify:

```
power m 3 = m * (m * m)
```

by using the reduction rules for `power`, *, and +.

Partial evaluation of types

In Martin-Löf type theory we can define the type-valued function `Power` $A\ n = A^n$. *Set* is the type of small types - a universe:

```
Power :: Set -> Nat -> Set

Power A 0        = 1
Power A (Succ n) = A * (Power A n)

Power A n = rec 1 (\x y -> A * y) n
```

Let $n = 3$. Simplify:

```
Power A 3 = A * (A * (A * 1))
```

by using the reduction rules for `Power`. Can we simplify further?

Normalization during type-checking

To check that

```
(2007,(4,(12,()))) :: Power Nat 3
```

we need to normalize the type:

```
(2007,(4,(12,()))) :: Nat * (Nat * (Nat * 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:

$$\text{syntax} \underset{\downarrow}{\overset{[\![-]\!]}{\rightleftarrows}} \text{model}$$

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

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

Plan

- Martin-Löf type theory with one universe and untyped conversion (like Martin-Löf 1972 + η-rule). Syntax, reduction, normal forms, and inference rules.
- Normalization algorithms for terms and types:
  - $\mathrm{nbe}^{A}_{\Gamma} t = \downarrow^{[\![A]\!]_{\rho_\Gamma}}_{|\Gamma|} [\![t]\!]_{\rho_\Gamma}$
  - $\mathrm{Nbe}_{\Gamma} A = \Downarrow_{|\Gamma|} [\![A]\!]_{\rho_\Gamma}$
- Correctness of normalization algorithm for terms and types means decidability of equality:
  - If $\Gamma \vdash t, t' : A$ then $t =_{\beta\eta} t'$ iff $\mathrm{nbe}^{A}_{\Gamma} t \equiv \mathrm{nbe}^{A}_{\Gamma} t' \in Tm$.
  - If $\Gamma \vdash A, A'$ then $A =_{\beta\eta} A'$ iff $\mathrm{Nbe}_{\Gamma} A \equiv \mathrm{Nbe}_{\Gamma} A' \in Tm$.

Martin-Löf Type Theory

Types and terms with de Bruijn indices (types are terms - universe à la Russell)

$Tm \ni r, s, t, z, A, B$	$::=$	v_i	de Bruijn index
	$\mid$	λt	abstracting 0th variable
	$\mid$	$r\,s$	application
	$\mid$	$Zero$	natural number "0"
	$\mid$	$Succ\,t$	successor
	$\mid$	$Rec\,A\,z\,s\,t$	primitive recursion
	$\mid$	$\Pi A\,B$	dependent function type
	$\mid$	Nat	natural number type
	$\mid$	Set	universe

We can add other set constructors too: $\Sigma A\,B, A + B, 0, 1$, 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.

$$
\begin{array}{lcll}
(\lambda t)\, s & \longrightarrow & t[s] & (\beta\text{-}\lambda) \\
\lambda.(\Uparrow^1 t)\, v_0 & \longrightarrow & t & (\eta) \\[1ex]
\mathit{Rec}\, A\, z\, s\, \mathit{Zero} & \longrightarrow & z & (\beta\text{-}\mathit{Rec}\text{-}\mathit{Zero}) \\
\mathit{Rec}\, A\, z\, s\, (\mathit{Succ}\, r) & \longrightarrow & s\, r\, (\mathit{Rec}\, A\, z\, s\, r) & (\beta\text{-}\mathit{Rec}\text{-}\mathit{Succ})
\end{array}
$$

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

Judgement forms

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

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

$$\frac{}{\Gamma \vdash \mathit{Nat} : \mathit{Set}} \qquad \frac{\Gamma \vdash A : \mathit{Set} \qquad \Gamma, A \vdash B : \mathit{Set}}{\Gamma \vdash \Pi A B : \mathit{Set}}$$

well-formed types

$$\frac{\Gamma \vdash A : \mathit{Set}}{\Gamma \vdash A} \qquad \frac{\Gamma \vdash}{\Gamma \vdash \mathit{Set}} \qquad \frac{\Gamma \vdash A \qquad \Gamma, A \vdash B}{\Gamma \vdash \Pi A B}$$

and the type conversion rule:

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash A'}{\Gamma \vdash t : A'} \; A =_{\beta\eta} A'$$

There are also introduction and elimination rules for Π and *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:

$$\begin{array}{rcl} A, B, t, u & ::= & \Pi A B \mid Nat \mid Set \mid \lambda t \mid Zero \mid Succ\, t \mid s \\ s & ::= & v_i \mid s\, t \mid Rec\, A\, t\, u\, s \end{array}$$

"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":

$$\begin{array}{lcl} \mathrm{Pi} & : & \mathrm{D} \times [\mathrm{D} \to \mathrm{D}] \to \mathrm{D} \\ \mathrm{Nat} & : & \mathrm{D} \\ \mathrm{Set} & : & \mathrm{D} \end{array} \qquad \begin{array}{lcl} \mathrm{Lam} & : & [\mathrm{D} \to \mathrm{D}] \to \mathrm{D} \\ \mathrm{Zero} & : & \mathrm{D} \\ \mathrm{Succ} & : & \mathrm{D} \to \mathrm{D} \\ \mathrm{Ne} & : & TM_{\perp} \to \mathrm{D} \end{array}$$

where $TM = N \to Tm_Z$ (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

```
type Cxt = [Tm]
```

Normalization of a term wrt a type and a context:

```
nbe  :: Cxt -> Tm -> Tm -> Tm
```

Normalization of a type wrt a context

```
nbeT :: Cxt -> Tm -> Tm
```

Evaluation function

$$
\begin{array}{lcl}
[\![\_]\!]_{\_} & : & Tm \to [[N \to D] \to \mathrm{D}] \\
\\
[\![v_i]\!]_\rho & = & \rho(i) \\
[\![\lambda t]\!]_\rho & = & \mathrm{Lam}\,(d \mapsto [\![t]\!]_{\rho,d}) \\
[\![r\,s]\!]_\rho & = & [\![r]\!]_\rho \cdot [\![s]\!]_\rho \\
[\![Zero]\!]_\rho & = & \mathrm{Zero} \\
[\![Succ\,t]\!]_\rho & = & \mathrm{Succ}\,[\![t]\!]_\rho \\
[\![Rec\,A\,z\,s\,t]\!]_\rho & = & \mathrm{rec}\,(d \mapsto [\![A]\!]_{\rho,d})\,[\![z]\!]_\rho\,[\![s]\!]_\rho\,[\![t]\!]_\rho \\
[\![\Pi A\,B]\!]_\rho & = & \mathrm{Pi}\,[\![A]\!]_\rho\,(d \mapsto [\![B]\!]_{\rho,d}) \\
[\![Nat]\!]_\rho & = & \mathrm{Nat} \\
[\![Set]\!]_\rho & = & \mathrm{Set}
\end{array}
$$

Application of normal forms in hoas

We define application on D as the function

$$
\begin{array}{lcll}
\mathrm{app} : [\mathrm{D} \longrightarrow [\mathrm{D} \longrightarrow \mathrm{D}]] & & & \\
(\mathrm{Lam}\, f) \cdot d & = & f\, d & \\
e \cdot d & = & \perp & \text{if } e \text{ is not } \mathrm{Lam}\, f
\end{array}
$$

where in the following "default $\perp$ clauses" like the last one are always tacitly assumed.
In Haskell:

```
appD :: D -> D -> D
appD (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

$$\Downarrow \,:\, [\mathrm{D} \longrightarrow TM_{\perp}]$$

$$\begin{array}{lcl}
\Downarrow_k (\mathrm{Pi}\, a\, g) & = & \Pi(\Downarrow_k a)\,(\Downarrow_{k+1} g\,(\uparrow^a \hat{v}_{-(k+1)})) \\
\Downarrow_k \mathrm{Nat} & = & \mathit{Nat} \\
\Downarrow_k \mathrm{Set} & = & \mathit{Set} \\
\Downarrow_k (\mathrm{Ne}\, \hat{t}) & = & \hat{t}(k)
\end{array}$$

$$\downarrow \,:\, [\mathrm{D} \longrightarrow [\mathrm{D} \longrightarrow TM_{\perp}]]$$

$$\begin{array}{lcl}
\downarrow_k^{\mathrm{Set}} a & = & \Downarrow_k a \\
\downarrow_k^{\mathrm{Pi}\, a\, g} (\mathrm{Lam}\, f) & = & \lambda(\downarrow_{k+1}^{g\,(\uparrow^a \hat{v}_{-(k+1)})} (f\,(\uparrow^a \hat{v}_{-(k+1)}))) \\
\downarrow_k^{\mathrm{Nat}} \mathrm{Zero} & = & \mathit{Zero} \\
\downarrow_k^{\mathrm{Nat}} (\mathrm{Succ}\, d) & = & \mathit{Succ}\,(\downarrow_k^{\mathrm{Nat}} d) \\
\downarrow_k^{c} (\mathrm{Ne}\, \hat{t}) & = & \hat{t}(k) \qquad \text{if } c \neq \perp, c \neq \mathrm{Pi} \ldots
\end{array}$$

Reflection

Mapping neutral terms (including variables) to D:

$$\uparrow : [\mathsf{D} \to [TM_\perp \to \mathsf{D}]]$$

$$\begin{aligned} \uparrow^{\mathsf{Pi}\, a\, g} \hat{t} &= \mathsf{Lam}\,(d \mapsto \uparrow^{g(d)}(\hat{t} \downarrow^{a} d)) \\ \uparrow^{c} \hat{t} &= \mathsf{Ne}\, \hat{t} & \text{if } c \neq \perp, c \neq \mathsf{Pi}\ldots \end{aligned}$$

We perform η-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

$$\mathrm{rec} : [[\mathrm{D} \to \mathrm{D}] \to [\mathrm{D} \to [\mathrm{D} \to [\mathrm{D} \to \mathrm{D}]]]]$$

$$\begin{array}{lcl}
\mathrm{rec}\, a\, d_z\, d_s\, \mathrm{Zero} & = & d_z \\
\mathrm{rec}\, a\, d_z\, d_s\, (\mathrm{Succ}\, e) & = & d_s \cdot e \cdot (\mathrm{rec}\, a\, d_z\, d_s\, e) \\
\mathrm{rec}\, a\, d_z\, d_s\, (\mathrm{Ne}\, \hat{t}) & = & \uparrow^{a(\mathrm{Ne}\, \hat{t})} (k \mapsto \mathit{Rec}\, (\Downarrow_{k+1} a(\mathrm{Ne}\, v_{-(k+1)})) \\
& & \qquad (\downarrow_k^{a\,\mathrm{Zero}} d_z) \\
& & \qquad (\downarrow_k^{\Pi\,\mathrm{Nat}\,(d \mapsto a\, d \Rightarrow a\,(\mathrm{Succ}\, d))} d_s) \\
& & \qquad \hat{t}(k))
\end{array}$$

Here we use reification $\downarrow$ and reflection $\uparrow$.

The normalization function

Normalization by evaluation for terms and types is now implemented by these two functions:

$$\begin{array}{lcl} \mathrm{nbe}^{A}_{\Gamma} t & := & \downarrow^{[\![A]\!]_{\rho_\Gamma}}_{|\Gamma|} [\![t]\!]_{\rho_\Gamma} \\ \mathrm{Nbe}_{\Gamma} A & := & \Downarrow_{|\Gamma|} [\![A]\!]_{\rho_\Gamma} \end{array}$$

where ρ_Γ 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 $\mathrm{nbe}^A_\Gamma t \equiv \mathrm{nbe}^A_\Gamma t' \in Tm$.
- If $\Gamma \vdash A, A'$ then $A =_{\beta\eta} A'$ iff $\mathrm{Nbe}_\Gamma A \equiv \mathrm{Nbe}_\Gamma A' \in Tm$.

We split it up into two parts

Completeness

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

Soundness

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

We will only discuss the former. The latter is shown by defining a Kripke logical relation between terms and their normal forms in hoas.

PER of natural numbers and PER of functions

We inductively define $\mathcal{N}at \in \mathrm{Per}$ by the following rules.

$$\frac{}{\mathsf{Zero} = \mathsf{Zero} \in \mathcal{N}at} \qquad \frac{d = d' \in \mathcal{N}at}{\mathsf{Succ}\, d = \mathsf{Succ}\, d' \in \mathcal{N}at} \qquad \frac{}{\mathsf{Ne}\, \hat{t} = \mathsf{Ne}\, \hat{t} \in \mathcal{N}at}$$

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:

$$\Pi \mathcal{A}\, \mathcal{G} = \{(e, e') \mid (e \cdot d, e' \cdot d') \in \mathcal{G}(d) \text{ for all } (d, d') \in \mathcal{A}\}.$$

Inductive-recursive definition of PER of small types

We simultaneously define the PER $\mathcal{S}et \in \mathrm{Rel}$ and the family of PERS $[a]$ for a in the domain of $\mathcal{S}et$ by the following rules.

$$\frac{a = a' \in \mathcal{S}et \qquad g(d) = g'(d') \in \mathcal{S}et \text{ for all } d = d' \in [a]}{\mathrm{Pi}\, a\, g = \mathrm{Pi}\, a'\, g' \in \mathcal{S}et}$$

$$\frac{}{\mathrm{Nat} = \mathrm{Nat} \in \mathcal{S}et} \qquad \frac{}{\mathrm{Ne}\,\hat{t} = \mathrm{Ne}\,\hat{t} \in \mathcal{S}et}$$

$$\begin{array}{rcl} [\mathrm{Pi}\, a\, g] & = & \Pi\, [a]\, (d \mapsto [g(d)]) \\ [\mathrm{Nat}] & = & \mathcal{N}at \\ [\mathrm{Ne}\,\hat{t}] & = & \mathcal{N}e. \end{array}$$

Inductive-recursive definition as monotone inductive definition

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

$$\frac{(a, \mathcal{A}) \in \mathsf{T} \quad (g(d), \mathcal{G}(d)) \in \mathsf{T} \text{ for all } d \in \mathcal{A}}{(\mathsf{Pi}\, a\, g, \Pi \mathcal{A}\, \mathcal{G}) \in \mathsf{T}}$$

$$\frac{}{(\mathsf{Nat}, \mathcal{N}at) \in \mathsf{T}} \qquad \frac{}{(\mathsf{Ne}\, \hat{t}, \mathcal{N}e) \in \mathsf{T}}$$

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:

$$\frac{c = c' \in \mathcal{S}et}{c = c' \in \mathcal{T}ype} \qquad \frac{}{\mathrm{Set} = \mathrm{Set} \in \mathcal{T}ype}$$

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

$$\begin{array}{rcl} [\mathrm{Pi}\, a\, g] & = & \Pi\, [a]\, (d \mapsto [g(d)]) \\ [\mathrm{Nat}] & = & \mathcal{N}at \\ [\mathrm{Ne}\, \hat{t}] & = & \mathcal{N}e. \\ [\mathrm{Set}] & = & \mathcal{S}et \end{array}$$

Reification and reflection preserve equality

1. If $c = c' \in \mathcal{T}ype$ then $\uparrow^{c} \hat{t} = \uparrow^{c'} \hat{t} \in [c]$.
2. If $c = c' \in \mathcal{T}ype$ then $\Downarrow c \equiv \Downarrow c' \in TM$.
3. If $c = c' \in \mathcal{T}ype$ 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 $A =_{\beta\eta} A'$ then $\Gamma \models A = A'$.
- If $\Gamma \vdash t, t' : A$ and $t =_{\beta\eta} t'$ then $\Gamma \models t = t' : A$.

where

$$\begin{aligned}
\Gamma \models A = A' \quad &:\Longleftrightarrow \quad \Gamma \models \text{ and } \forall \rho = \rho' \in [\Gamma].\ [\![A]\!]_\rho = [\![A']\!]_{\rho'} \in \mathcal{T}ype \\
\Gamma \models t = t' : A \quad &:\Longleftrightarrow \quad \Gamma \models A \text{ and } \forall \rho = \rho' \in [\Gamma].\ [\![t]\!]_\rho = [\![t']\!]_{\rho'} \in [[\![A]\!]_\rho]
\end{aligned}$$

Completeness of NbE

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

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.
- Cf work by Martin-Löf 1973, 2004. Also work by Danielsson 2006.
- Key obstacle was overcome by starting with untyped nbe. (Note also that the algorithm for MLTT with only beta-conversion is more straightforward.)
- Future work. Equality judgments (LiCS 2007). Cwfs. Correctness of type-checking. Meta-theorems.