Normalization by Evaluation in the Delay Monad

Andreas Abel[1] James Chapman [2]

[1]Department of Computer Science and Engineering
Gothenburg University, Sweden

[2]Computer and Information Sciences
Strathclyde University

Types for Proofs and Programs
Novi Sad, Serbia
24 May 2016

A posteri normalization

- Implementing partiality (potential non-termination) in a total language
- Case study: Normalization by Evaluation (NbE) for the simply-typed lambda calculus
- STL is normalizing, so we could define evaluation by recursion on the termination proof
- This work:
  1. define evaluation as partial function
  2. show its correctness w.r.t. an equalitional theory
  3. optionally: show termination
- Stress-test for the new coinduction (copatterns and sized types) in Agda.

Simply-Typed Lambda Terms and Values

```
data Tm (Γ : Cxt) : (a : Ty) → Set where
  var  : ∀{a}   (x : Var Γ a)                    → Tm Γ a
  abs  : ∀{a b} (t : Tm (Γ , a) b)               → Tm Γ (a ⇒ b)
  app  : ∀{a b} (t : Tm Γ (a ⇒ b)) (u : Tm Γ a)  → Tm Γ b

mutual

  data Val : (a : Ty) → Set where
    lam : ∀{Γ a b} (t : Tm (Γ , a) b) (ρ : Env Γ)  → Val (a ⇒ b)

  data Env : (Γ : Cxt) → Set where
    ε    : Env ε
    _,_  : ∀{Γ a} (ρ : Env Γ) (v : Val a) → Env (Γ , a)
```

A Functional Call-By-Value Interpreter

Evaluator (draft).

```
mutual
  ⟦_⟧_ : ∀{Γ a} → Tm Γ a → Env Γ → Val a
  ⟦ var x     ⟧ ρ = lookup x ρ
  ⟦ abs t     ⟧ ρ = lam t ρ
  ⟦ app r s   ⟧ ρ = apply (⟦ r ⟧ ρ) (⟦ s ⟧ ρ)

  apply : ∀{a b} → Val (a ⇒ b) → Val a → Val b
  apply (lam t ρ) v = ⟦ t ⟧ (ρ , v)
```

Of course, termination check fails!

Coinductive Delay

```
CoInductive Delay (A : Type) : Type :=
| return (a  : A)
| later  (a? : Delay A).
```

```
mutual
  data Delay (A : Set) : Set where
    return : (a  : A)          → Delay A
    later  : (a′ : ∞Delay A)   → Delay A

  record ∞Delay (A : Set) : Set where
    coinductive
    field force : Delay A
open ∞Delay public
```

The Coinductive Delay Monad

Nonterminating computation $\bot$.

$$\mathsf{forever} : \forall\{A\} \to \infty\mathsf{Delay}\ A$$
$$\mathsf{force\ forever} = \mathsf{later\ forever}$$

Monad instance.

```
mutual
  _»=_ : ∀{A B} → Delay A → (A → Delay B) → Delay B
  (return a  »= k)  =  k a
  (later a'  »= k)  =  later (a' ∞»= k)

  _∞»=_ : ∀{A B} → ∞Delay A → (A → Delay B) → ∞Delay B
  force (a' ∞»= k)  =  force a' »= k
```

Evaluation In The Delay Monad

Monadic evaluator.

$$\llbracket\_\rrbracket\_ : \forall\{\Gamma\ a\} \to \mathsf{Tm}\ \Gamma\ a \to \mathsf{Env}\ \Gamma \to \mathsf{Delay}\ (\mathsf{Val}\ a)$$
$$\llbracket\ \mathsf{var}\ x\ \rrbracket\ \rho = \mathsf{return}\ (\mathsf{lookup}\ x\ \rho)$$
$$\llbracket\ \mathsf{abs}\ t\ \rrbracket\ \rho = \mathsf{return}\ (\mathsf{lam}\ t\ \rho)$$
$$\llbracket\ \mathsf{app}\ r\ s\ \rrbracket\ \rho = \mathsf{apply}\ (\llbracket\ r\ \rrbracket\ \rho)\ (\llbracket\ s\ \rrbracket\ \rho)$$

$$\mathsf{apply} : \forall\{a\ b\} \to \mathsf{Delay}\ (\mathsf{Val}\ (a \Rightarrow b)) \to \mathsf{Delay}\ (\mathsf{Val}\ a) \to \mathsf{Delay}\ (\mathsf{Val}\ b)$$
$$\mathsf{apply}\ u?\ v? = u?\ \gg\!=\ \lambda\ u \to$$
$$v?\ \gg\!=\ \lambda\ v \to$$
$$\mathsf{later}\ (\infty\mathsf{apply}\ u\ v)$$

$$\infty\mathsf{apply} : \forall\{a\ b\} \to \mathsf{Val}\ (a \Rightarrow b) \to \mathsf{Val}\ a \to \infty\mathsf{Delay}\ (\mathsf{Val}\ b)$$
$$\mathsf{force}\ (\infty\mathsf{apply}\ (\mathsf{lam}\ t\ \rho)\ v) = \llbracket\ t\ \rrbracket\ (\rho\ ,\ v)$$

Productive? Not guarded by constructors!

Sized Coinductive Delay Monad

```
data Delay (i : Size) (A : Set) : Set where
  return : (a  : A)                → Delay i A
  later  : (a′ : ∞Delay i A)       → Delay i A

record ∞Delay (i : Size) (A : Set) : Set where
  coinductive
  field force : ∀{j : Size< i} → Delay j A
```

- Size = depth = how often can we force?
- Not to be confused with "number of laters"!

Sized Coinductive Delay Monad (II)

```
record ∞Delay i A : Set where
  coinductive
  field force : ∀{j : Size< i} → Delay j A
```

Corecursion = induction on depth.

```
forever : ∀{i A} → ∞Delay i A
force (forever {i}) {j} = later (forever {j})
```

Since $j < i$, the recursive call forever $\{j\}$ is justified.

Sized Coinductive Delay Monad (III)

Monadic bind preserves depth.

$$
\begin{array}{lll}
\_\gg\!=\_ & : & \forall\{i\ A\ B\} \to \\
 & & \mathsf{Delay}\ i\ A \to (A \to \mathsf{Delay}\ i\ B) \to \mathsf{Delay}\ i\ B \\
(\mathsf{return}\ a & \gg\!=\ k) & =\ k\ a \\
(\mathsf{later}\ a' & \gg\!=\ k) & =\ \mathsf{later}\ (a'\ \infty\!\gg\!=\ k)
\end{array}
$$

$$
\begin{array}{lll}
\_\infty\!\gg\!=\_ & : & \forall\{i\ A\ B\} \to \\
 & & \infty\mathsf{Delay}\ i\ A \to (A \to \mathsf{Delay}\ i\ B) \to \infty\mathsf{Delay}\ i\ B \\
\mathsf{force}\ (a'\ \infty\!\gg\!=\ k) & & =\ \mathsf{force}\ a'\ \gg\!=\ k
\end{array}
$$

Depth of $a?\ \gg\!=\ k$ is at least minimum of depths of $a?$ and $k\ a$.

Sized Corecursive Evaluator

Add sizes to type signatures.

$$[\![\_]\!]\_ : \forall\{i\ \Gamma\ a\} \to \mathsf{Tm}\ \Gamma\ a \to \mathsf{Env}\ \Gamma \to \mathsf{Delay}\ i\ (\mathsf{Val}\ a)$$

$$\begin{array}{l} \mathsf{apply} : \forall\{i\ a\ b\} \to \\ \quad \mathsf{Delay}\ i\ (\mathsf{Val}\ (a \Rightarrow b)) \to \mathsf{Delay}\ i\ (\mathsf{Val}\ a) \to \mathsf{Delay}\ i\ (\mathsf{Val}\ b) \\ \mathsf{apply}\ u?\ v? = \ u?\ \gg\!= \lambda\ u\ \to \\ \qquad\qquad\qquad\ v?\ \gg\!= \lambda\ v\ \to \\ \qquad\qquad\qquad\ \mathsf{later}\ (\infty\mathsf{apply}\ u\ v) \end{array}$$

$$\begin{array}{l} \infty\mathsf{apply} : \forall\{i\ a\ b\} \to \mathsf{Val}\ (a \Rightarrow b) \to \mathsf{Val}\ a \to \infty\mathsf{Delay}\ i\ (\mathsf{Val}\ b) \\ \mathsf{force}\ (\infty\mathsf{apply}\ (\mathsf{lam}\ t\ \rho)\ v) = [\![\ t\ ]\!]\ (\rho\ ,\ v) \end{array}$$

Termination checker is happy!

Normalization by Evaluation (preliminary)

- Add neutrals (variables applied to normal forms) to values.
- Read back values into normal forms.
  Function values are applied to a (fresh) variable.

$$\mathsf{readback} : \forall\{i\ \Gamma\ a\} \to \mathsf{Val}\ i\ \Gamma\ a \to \mathsf{Delay}\ i\ (\mathsf{Nf}\ \Gamma\ a)$$

- Normalization is evaluation followed by readback.

$$\mathsf{idenv} : \forall\{i\ \Gamma\} \to \mathsf{Env}\ i\ \Gamma\ \Gamma$$

$$\mathsf{nf} : \forall\{i\ \Gamma\ a\}(t : \mathsf{Tm}\ \Gamma\ a) \to \mathsf{Delay}\ i\ (\mathsf{Nf}\ \Gamma\ a)$$
$$\mathsf{nf}\ t = \mathsf{readback}\ (\mathsf{eval}\ t\ \mathsf{idenv})$$

Completeness of NbE

- Typed $\beta\eta$-equality $\Gamma \vdash t = t' : a$.

$$\frac{\Gamma, x{:}a \vdash r : b \qquad \Gamma \vdash s : a}{\Gamma \vdash (\lambda x r)\, s = r[s/x] : b}$$

- Normalization of $\beta\eta$-equal terms should be weakly bisimilar.
- Our monadic cbv-evaluation does not model cbn-β.

$$[\![(\lambda x r)\, s]\!]_\rho = [\![r]\!]_{\rho, [\![s]\!]_\rho} \stackrel{?}{=} [\![r[s/x]]\!]_\rho$$

  The effects of evaluating s come too early.
- We need a lazier evaluator.

Lazy Values

- We fuse the delay monad into the value type.
- Values are now coinductive.

```
data Val (i : Size) (Δ : Cxt) : (a : Ty) → Set where
  lam   : ∀{Γ a b}  (t    : Tm (Γ , a) b)
                    (ρ    : Env i Δ Γ)      → Val i Δ (a ⇒ b)
  later : ∀{a}      (v∞   : ∞Val i Δ a)     → Val i Δ a
  ne    : ∀{a}      (n    : NeVal i Δ a)    → Val i Δ a

record ∞Val (i : Size) (Δ : Cxt) (a : Ty) : Set where
  coinductive
  field force : {j : Size< i} → Val j Δ a
```

- The neutrals are for reification.

Lazy Evaluation

```
eval : ∀{i Γ Δ a} → Tm Γ a → Env i Δ Γ → Val i Δ a
eval (app t u)  ρ = apply (eval t ρ) (eval u ρ)

apply : ∀ {i Δ a b} → Val i Δ (a ⇒ b) → Val i Δ a → Val i Δ b
apply (ne w)      v = ne (app w v)
apply (lam t ρ)  v = later (beta t ρ v)
apply (later w)  v = later (∞apply w v)

∞apply : ∀ {i Δ a b} → ∞Val i Δ (a ⇒ b) → Val i Δ a → ∞Val i Δ b
force (∞apply w v) = apply (force w) v

beta : ∀ {i Γ a b} (t : Tm (Γ , a) b)
   {Δ : Cxt} (ρ : Env i Δ Γ) (v : Val i Δ a) → ∞Val i Δ b
force (beta t ρ v) = eval t (ρ , v)
```

Readback

```
readback : ∀{i Γ a} → Val i Γ a → Delay i (Nf Γ a)
readback {a = *} (ne w)      = ne  <$> nereadback w
readback {a = *} (later w)   = later (∞readback w)
readback {a = _ ⇒ _} v       = later (abs ∞<$> eta v)

∞readback : ∀{i Γ a} → ∞Val i Γ a → ∞Delay i (Nf Γ a)
force (∞readback w) = readback (force w)

eta : ∀{i Γ a b} → Val i Γ (a ⇒ b) → ∞Delay i (Nf (Γ , a) b)
force (eta v) = readback (apply (weakVal v) (ne (var zero)))

nereadback : ∀{i Γ a} → NeVal i Γ a → Delay i (Ne Γ a)
nereadback (var x)     = return (var x)
nereadback (app w v)   = app <$> nereadback w <*> readback v
```

Completeness Proof

- Logical relation on values for completeness:

$$
\begin{array}{lcll}
[\![\star]\!]^{\Gamma} \; (v, v') & = & \mathsf{readback}\, v \sim \mathsf{readback}\, v' & \text{weakly bisimilar} \\
[\![a \Rightarrow b]\!]^{\Gamma} \; (f, f') & = & \forall \eta \in \mathsf{Ren}\,\Delta\,\Gamma,\; [\![a]\!]^{\Delta}(u, u') \Longrightarrow [\![b]\!]^{\Delta}(f\,\eta\,u,\; f'\,\eta\,u') &
\end{array}
$$

- Fundamental theorem:
  If $\Gamma \vdash t = t' : a$ and $[\![\Gamma]\!]^{\Delta}(\rho, \rho')$ then $[\![a]\!]^{\Delta}([\![t]\!]_{\rho}, [\![t']\!]_{\rho'})$.

Conclusions

- Agda's new coinduction gives us flexibility in corecursive definitions.
- Don't be scared of sized types!
- We can do meta theory of partial STL!
- Applicable to Type:Type ?

Related Work

- Danielsson, Operational Semantics Using the Partiality Monad (ICFP 2012)
- Leroy, Gregoire A Compiled Implementation of Strong Reduction (ICFP 2002)