Normalization by Evaluation in the Delay Monad: An Extended Case Study for Coinduction Via Copatterns and Sized Types

Andreas Abel[1]*and James Chapman[2]†

[1] Institute of Cybernetics, Tallinn
[2] Gothenburg University

Abstract

In this paper, we present an Agda formalization of a normalizer for simply-typed lambda terms and its accompanying normalization proof. The normalizer consists of two coinductively defined functions in the delay monad: One is a standard evaluator of lambda terms to closures, the other a type-directed reifier from values to eta-long beta-normal forms. Their composition, normalization-by-evaluation, is shown to be a total function a posteriori, using a standard logical-relations argument. This paper builds on our workshop paper[Abel and Chapman, 2014] which presented only the normalizer and termination proof. Here we also show soundness and completeness of the normalizer, thus completing the normalization proof. The complete formalization serves as a proof-of-concept for coinductive programming and reasoning using sized types and copatterns, a new and presently experimental feature of Agda.

1 Introduction

In dependently typed programming languages such as Agda [AgdaTeam, 2014], recursively defined functions must be *structurally* recursive. This requirement has the effect of fusing the program and its termination proof. It is very convenient if the function is naturally structurally recursive as there is no extra proof effort. However, it is inconvenient when the structure on which the function is structurally recursive is not visible or its comprehension is beyond the capabilities of the termination checker. Whilst there is nothing wrong with writing algorithms that are a priori structurally recursive (indeed, it should be encouraged), we would also like to support a more step-by-step process of development where the user can write an program that is not structural and prove termination later. Yet languages based on the Curry-Howard-Isomorphism, such as Coq [INRIA, 2012] and Agda, forbid definitions that do not denote total functions, as they can be exploited to prove false theorems. However, rather than writing a *terminating* recursive function in the first instance, one can write a *productive* corecursive function. A potential recursive function $f : A \to B$ that would be rejected by the termination checker can be written as a corecursive function $f : A \to \mathsf{Delay}\, B$, where the type $\mathsf{Delay}\, B$ represents values that can arrive after a possibly infinite delay [Capretta, 2005]. Showing termination amounts to proving that such a delay is in fact only finite: $\exists b.\ f\, a \Downarrow b$. By combining the corecursive function and its termination proof one gets a terminating recursive function. In this paper we demonstrate this approach on an canonical normalization-by-evaluation (NBE) style normalizer for simply typed lambda calculus that is not structurally recursive. Its main difference from standard NBE is that it uses first-order closures as semantic values.

*Andreas Abel acknowledges support by Vetenskapsrådet grant 254820104 given to Thierry Coquand.

†James Chapman has been supported by the ERDF funded Estonian CoE project EXCS and ICT National Programme project "Coinduction", the Estonian Ministry of Research and Education target-financed research theme no. 0140007s12 and the Estonian Science Foundation grant no. 9219.

2 Delay Monad

The type Delay A of possibly non-terminating computations of type A is the greatest fixed-point of the functor $F(X) = A+X$. In Agda, the Delay type can be represented as a mutual definition of an inductive datatype and a coinductive record. The record ∞Delay is a coalgebra and one interacts with it by using its single observation (copattern) force. Once forced we get an element of the Delay datatype which we can pattern match on to see if the value is available now or later. If it is later, then we get an element of ∞Delay which we can force again, and so forth. In Agda syntax, this is expressed as follows:

```
mutual  data Delay (A : Set) : Set where
          now   : A          → Delay A
          later : ∞Delay A   → Delay A

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

We define convergence $a? \Downarrow a$ as a relation between delayed computations of type Delay A and values of type A. If $a? \Downarrow a$, then the delayed computation $a?$ eventually yields the value a. This is a central concept in this paper, as we will write a (productive) normalizer that produces delayed normal forms and then prove that all such delayed normal forms converge to a value, yielding termination of the normalizer. Notice that convergence is an *inductive* relation defined on coinductive data.

```
data _⇓_ {A : Set} : (a? : Delay A) (a : A) → Set where
  now⇓  : ∀{a}                                  → now a    ⇓ a
  later⇓ : ∀{a} {a∞ : ∞Delay A} → force a∞ ⇓ a  → later a∞ ⇓ a
```

3 Normalization

We construct a normalization function nf that takes a well-typed term t and returns a potentially delayed normal form.

```
nf : ∀{Γ a}(t : Tm Γ a) → Delay (Nf Γ a)
```

The termination proof states that for any term t, there exists a normal form n such that the result of normalization nf t converges to n after a finite delay.

```
normalize : ∀ {Γ a} (t : Tm Γ a) → ∃ λ n → nf t ⇓ n
```

Soundness of the normalizer states that any two terms t and t' that are related in the equational theory $t \sim t'$ have equal normal forms. The normal form n is obtained as the existential witness (first projection fst) of a *run* of the normalization proof normalize t.

```
sound : ∀ Γ a {t t' : Tm Γ a} → t ~ t' → fst (normalize t) ≡ fst (normalize t')
```

The main work in the completeness proof is to conclude that any term t is related in the equational theory to its normal form.

```
complete-lemma : ∀ Γ a (t : Tm Γ a) → t ~ emb (fst (normalize t))
```

Having proved soundness and complete-lemma, completeness follows as a corollary.

```
complete : ∀ Γ a {t t' : Tm Γ a} → fst (normalize t) ≡ fst (normalize t') → t ~ t'
```

4 Conclusion

Our case study suggests that encoding partial functions via the delay monad is a viable alternative to encoding via inductive functional relations even for non-trivial functions such as an evaluator. In future work, we would like to investigate whether the technique is beneficial in the daunting task of a formalized semantics of dependent type theory [Chapman, 2009].

References

Andreas Abel and James Chapman. Normalization by evaluation in the delay monad: A case study for coinduction via copatterns and sized types. In Paul Levy and Neel Krishnaswami, editors, Proceedings 5th Workshop on *Mathematically Structured Functional Programming,* Grenoble, France, 12 April 2014, volume 153 of *Electronic Proceedings in Theoretical Computer Science*, pages 51–67. Open Publishing Association, 2014.

AgdaTeam. The Agda Wiki, 2014. URL `http://wiki.portal.chalmers.se/agda`.

Venanzio Capretta. General recursion via coinductive types. *Logical Methods in Computer Science*, 1(2), 2005.

James Chapman. Type theory should eat itself. *Electronic Notes in Theoretical Computer Science*, 228:21–36, 2009. Proceedings of the International Workshop on Logical Frameworks and Metalanguages: Theory and Practice (LFMTP 2008).

INRIA. *The Coq Proof Assistant Reference Manual*. INRIA, version 8.4 edition, 2012. URL `http://coq.inria.fr/`.