Friends with Benefits

Implementing Foundational Corecursion in Isabelle/HOL (Extended Abstract)

Jasmin Christian Blanchette[1,2], Aymeric Bouzy[3], Andreas Lochbihler[4], Andrei Popescu[5], and Dmitriy Traytel[4]

[1] Inria & LORIA, Nancy, France
[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[3] Laboratoire d'informatique, École polytechnique, Palaiseau, France
[4] Institute of Information Security, ETH Zürich, Switzerland
[5] School of Science and Technology, Middlesex University, UK

Abstract. We describe AmiCo, a tool that extends Isabelle/HOL with flexible function definitions well beyond primitive corecursion. All definitions are certified by the assistant's inference kernel to guard against inconsistencies. A central notion is that of *friends*: functions that preserve the productivity of their arguments and that may occur in corecursive call contexts. As new friends are registered, corecursion benefits by becoming more expressive.

1 Introduction

Codatatypes and corecursion [8, 10, 15] are emerging as a major methodology for programming with infinite objects. Unlike in traditional lazy functional programming, codatatypes support *total* (*co*)*programming* [1, 3, 19], where the defined functions have a straightforward set-theoretic semantics and productivity is guaranteed. The proof assistants Agda [7], Coq [4], and Matita [2] have been supporting and promoting this methodology for years.

By contrast, proof assistants based on higher-order logic (HOL), such as HOL4 [16], HOL Light [11], and Isabelle/HOL [13, 14], have traditionally provided only datatypes. Isabelle/HOL is the first of these systems to offer codatatypes. It took two years, and about 24 000 lines of Standard ML, to move from an understanding of the mathematics [18] to an implementation that automates the process of checking high-level user specifications and producing the necessary corecursion and coinduction theorems [5].

There are important differences between Isabelle/HOL and type theory systems such as Coq in the way they currently handle corecursion. Consider the codatatype of streams given by

$$\mathtt{codatatype}\ \alpha\ \mathsf{stream} = (\mathsf{hd}\colon \alpha) \lhd (\mathsf{tl}\colon \alpha\ \mathsf{stream})$$

where $\lhd$ is the constructor and hd, tl are the selectors. In Coq, a definition such as

$$\begin{array}{l}\mathtt{corec}\ \mathsf{natsFrom} : \mathsf{nat} \to \mathsf{nat}\ \mathsf{stream}\ \mathtt{where}\\ \quad \mathsf{natsFrom}\ n = n \lhd \mathsf{natsFrom}\ (n+1)\end{array}$$

which introduces the function $n \mapsto n \lhd n+1 \lhd n+2 \lhd \cdots$, is accepted after a syntactic check that detects the $\lhd$-guardedness of the corecursive call. In Isabelle, this check is replaced by a deeper analysis. The `primcorec` command [5] transforms a user specification into a *blueprint* object: the coalgebra $b = \lambda n.\ (n, n+1)$. Then natsFrom is defined as $\mathsf{corec}_{\mathsf{stream}}\ b$, where $\mathsf{corec}_{\mathsf{stream}}$ is the fixed primitive corecursive combinator associated with the codatatype α stream. Finally, the user specification is is derived as a theorem from the characteristic equation of the corecursor.

Unlike in type theories, where (co)datatypes and (co)recursion are built-in, the HOL philosophy is to reduce every new construction to the core logic. This usually requires a lot of implementation work but guarantees that definitions introduce no inconsistencies. Isabelle's approach is admittedly more bureaucratic than Coq's, but for end users the net effect is the same: They obtain their specification as a theorem.

Since codatatypes, corecursion, and coinduction are derived concepts, there is no a priori restriction on the expressiveness of user specifications other than the expressiveness of HOL itself. Consider a variation of the function natsFrom, where addOne : nat $\to$ nat stream $\to$ nat stream is a function that adds one to each element of a stream:

```
corec natsFrom : nat → nat stream where
  natsFrom n = n ◁ addOne (natsFrom n)
```

Coq's syntactic check rejects this definition, because it does not know what to do about addOne. After all, addOne could explore the tail of its argument or beyond, hence blocking productivity and leading to underspecification or even an inconsistency.

Isabelle's additional bookkeeping allows for more nuances. Suppose addOne has been defined corecursively:

```
corec addOne : nat stream → nat stream where
  addOne n = (hd n + 1) ◁ addOne (tl n)
```

When analyzing addOne's specification, the `corec` command synthesizes its definition as a blueprint b. This definition can then be proved to be *friendly*, hence acceptable in corecursive call contexts when defining other functions. Functions with friendly definitions are called *friendly*, or *friends*. Intuitively, a function is friendly if it consumes at most one constructor before producing at least one.

In previous work [6], we presented the category theory underlying friends, including its connection to relational parametricity. We now introduce a tool, AmiCo, that automates the process of applying and incrementally improving corecursion by synthesizing and manipulating friends.

To demonstrate AmiCo's expressive power and convenience, we used it to formalize seven case studies in Isabelle, featuring a variety of codatatypes. Most of these are a good fit for our friend-based approach; a few required ingenuity and suggest directions for future work. The most convincing example relies on *self-friendship*, a concept introduced by Blanchette et al.; until now, no realistic applications were known of such "narcissist" definitions.

At the low level, the *corecursion state* summarizes what the system knows at a given point, including the set of available friends and a corecursor *up to* friends. Polymorphism complicates the picture, because some friends may only be available for specific

instances of a polymorphic codatatype. To each corecursor corresponds a coinduction principle up to friends and a uniqueness theorem that can be used to reason about corecursive functions.

All of the constructions and theorems derived from first principles, without requiring new axioms or extensions of the logic. This *foundational approach* prevents the introduction of inconsistencies, such as those that have affected the termination and productivity checkers of Agda [17] and Coq [9].

The user interacts with our tool via proof assistant commands. The `corec` command defines a function `f` by extracting a blueprint *b* from a user's specification, defining `f` using *b* and some corecursor, and deriving the original specification from the characteristic property of the corecursor. Specifying the `friend` option to `corec` additionally registers the function under definition as a friend, enriching the corecursor state. Moreover, `corec` supports mixed recursion–corecursion specifications, exploiting existing proof assistant infrastructure for terminating (well-founded) recursion [12]. Semantic proof obligations that must be discharged, notably termination, are either proved automatically or presented to the user. Another command, `friend_of_corec`, registers existing functions as friendly. Friend registration is an instance of term inferences directed by logical relations, combined with parametricity proofs.

AmiCo is a significant piece of engineering, at about 7 000 lines of Standard ML code. The tool is available as part of the development version of Isabelle and is scheduled for inclusion in the next official release.

We refer to our conference submission for details.[6]

References

[1] Abbott, M., Altenkirch, T., Ghani, N.: Containers: Constructing strictly positive types. Theor. Comput. Sci. 342(1), 3–27 (2005)

[2] Asperti, A., Ricciotti, W., Coen, C.S., Tassi, E.: The Matita interactive theorem prover. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE-23. LNCS, vol. 6803, pp. 64–69. Springer (2011)

[3] Atkey, R., McBride, C.: Productive coprogramming with guarded recursion. In: Morrisett, G., Uustalu, T. (eds.) ICFP 2013. pp. 197–208. ACM (2013)

[4] Bertot, Y., Casteran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Springer (2004)

[5] Blanchette, J.C., Hölzl, J., Lochbihler, A., Panny, L., Popescu, A., Traytel, D.: Truly modular (co)datatypes for Isabelle/HOL. In: Klein, G., Gamboa, R. (eds.) ITP 2014. LNCS, vol. 8558, pp. 93–110. Springer (2014)

[6] Blanchette, J.C., Popescu, A., Traytel, D.: Foundational extensible corecursion: a proof assistant perspective. In: ICFP 2015. pp. 192–204 (2015)

[7] Bove, A., Dybjer, P., Norell, U.: A brief overview of Agda—A functional language with dependent types. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 73–78. Springer (2009)

[8] Clouston, R., Bizjak, A., Grathwohl, H.B., Birkedal, L.: Programming and reasoning with guarded recursion for coinductive types. In: Pitts, A.M. (ed.) FoSSaCS 2015. LNCS, vol. 9034, pp. 407–421. Springer (2015)

[6] `http://www.loria.fr/~jablanch/amico.pdf`

[9] Dénès, M.: [Coq-Club] Propositional extensionality is inconsistent in Coq (2013), archived at https://sympa.inria.fr/sympa/arc/coq-club/2013-12/msg00119.html
[10] Giménez, E.: An application of co-inductive types in coq: Verification of the alternating bit protocol. In: Berardi, S., Coppo, M. (eds.) TYPES '95. LNCS, vol. 1158, pp. 135–152. Springer (1996)
[11] Harrison, J.: HOL Light: A tutorial introduction. In: FMCAD '96. LNCS, vol. 1166, pp. 265–269. Springer (1996)
[12] Krauss, A.: Partial recursive functions in higher-order logic. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS, vol. 4130, pp. 589–603. Springer (2006)
[13] Nipkow, T., Klein, G.: Concrete Semantics: With Isabelle/HOL. Springer (2014)
[14] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic. Springer (2002)
[15] Rutten, J.J.M.M.: Universal coalgebra: A theory of systems. Theor. Comput. Sci. 249, 3–80 (2000)
[16] Slind, K., Norrish, M.: A brief overview of HOL4. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) TPHOLs 2008. LNCS, vol. 5170, pp. 28–32. Springer (2008)
[17] Traytel, D.: [Agda] Agda's copatterns incompatible with initial algebras (2014), archived at https://lists.chalmers.se/pipermail/agda/2014/006759.html
[18] Traytel, D., Popescu, A., Blanchette, J.C.: Foundational, compositional (co)datatypes for higher-order logic—Category theory applied to theorem proving. In: LICS 2012, pp. 596–605. IEEE (2012)
[19] Turner, D.A.: Elementary strong functional programming. In: Hartel, P.H., Plasmeijer, M.J. (eds.) FPLE '95. LNCS, vol. 1022, pp. 1–13. Springer (1995)