Type-Based Methods for Termination and Productivity in Coq

Bruno Barras
INRIA Saclay
bruno.barras@inria.fr

Jorge Luis Sacchini
Carnegie Mellon University — Doha, Qatar
sacchini@qatar.cmu.edu

Coq is a *total* dependently-typed programming language: recursive functions must be terminating and co-recursive functions must be productive. The requirement of totality is essential to ensure logical consistency, since a non-terminating function can be easily used to encode a proof of falsity.

Systems based on dependent type theories, such as Coq and Agda, typically use syntactic methods, called *guard predicates* in Coq, to ensure termination (and productivity). A guard predicate is a form of static analysis, performed on the body of recursive functions, that checks that recursive functions are placed on structurally smaller arguments.

Guard predicates were initially implemented in Coq over 15 years ago by Eduardo Giménez. Throughout the years, the guard predicate implementation has been relaxed and extended, in order to accept more recursion patterns as terminating (the most recent addition involves commutative cuts [6]). As a result, the implementation is large and difficult to maintain, making the termination checker one of the weakest point in the Coq kernel. This is highly undesirable, since any bug at this level jeopardizes logical consistency. Furthermore, the metatheoretical properties of the implemented extensions have not been studied (in particular, logical consistency).

From the user point of view, the limitations of syntactic-based termination appear often in practice. Let us illustrate with a typical example. Consider the following definitions of subtraction and division on natural numbers, where `div` $x\,y$ computes $\lceil \frac{x}{y+1} \rceil$ by repeated subtraction:

```
Fixpoint minus x y { struct x } :=
  match x, y with
  | O, _ => x
  | S x', O => x
  | S x', S y' => minus x' y' end.
```

```
Fixpoint div x y { struct x } :=
  match x with
  | O => O
  | S x' => S (div (minus x' y) y) end.
```

Coq accepts both functions as terminating. However, in the case of `div`, this depends greatly on how `minus` is defined: if we define `minus O y = O`, while it would not affect its behavior, `div` would no longer satisfy the guard predicate.

This example illustrates an important limitation of guard predicates: lack of compositionality. For example, in the case above, the *code* of `minus` must be available for `div`. This prevents the use of higher-order functions and hinders modular design.

Another disadvantage of this approach is performance. The guard predicate is checked on the normal form of the body which is usually much larger (in the case of `div` it forces the unfolding of `minus` and β-reduction on `x'` and `y`). This has a significant impact in the time needed to typecheck large specifications. Furthermore, it is difficult to provide meaningful errors messages, since the guard predicate is not checked on the code the user provided.

The limitations of using syntactic-based methods for ensuring termination have been recognized for some time. One long-studied alternative, that provides a better balance between expressive power, ease of implementation, and ease of understanding, is *type-based termination* (see e.g. [1–3, 5, 7]).

The core idea behind the type-based termination approach is the use of *sized types*, i.e. types of the form T^s where T is a (co-)inductive type and s is a size annotation which represents an upper bound on the size of the type elements. Size information is used to track the size of arguments in recursive calls. Termination of recursive functions is ensured by restricting recursive calls to be performed only on smaller arguments, as shown by their type.

The main advantage of type-based methods against syntactic-based methods is that size information is exported in the type of a function. For example, when typechecking `div`, only the type of `minus` is

needed. (`minus` has type $\texttt{nat}^s \to \texttt{nat} \to \texttt{nat}^s$, which means that the size of the result is not bigger than the size of the first argument. Any function with this type can be used to type `div`.) In other words, type-based methods are compositional, enabling modular design of specifications.

Another advantage of having size information in the type of a function is that it allows non-structural recursion; a typical example is the `quicksort` function (see e.g. [4]). In general, type-based methods are strictly more expressive than syntactic-based methods.

The notion of sized types is semantically intuitive, as it corresponds directly to the interpretation of (co-)inductive types as the least (greatest) fixed point of a monotone operator. The least fixed point is known to be reached by transfinite iteration of the operator at some ordinal. Sizes can be interpreted as ordinals and sized types represent approximations of the operator. One consequence of this fact is that soundness (i.e. strong normalization and logical consistency) is relatively easy to establish—and has already been established for several theories including the Calculus of Inductive Constructions.

Furthermore, type-based methods treat termination and productivity uniformly. In Coq, guard predicates for checking productivity use a different (although, somewhat dual) procedure where co-recursive call are only permitted under a constructor (the predicate is called *guarded-by-constructor* while in the case of termination, the predicate is called *guarded-by-destructor*). Syntactic methods for ensuring productivity suffer from the same limitations as in the case of termination, while type-based methods offer similar advantages in terms of compositionality and expressive power, while at the same time permitting a uniform treatment of recursive and co-recursive definitions.

Given the current state of the implementation of termination and productivity checking in Coq, and the advantages that type-based methods can provide, we believe it is worthy to pursue their implementation in the Coq kernel. Such implementation must be done with extreme care, as termination checking is a critical component in the kernel and any modification will likely affect many components in the upper layers of the Coq architecture.

In this proposed talk, we will discuss the advantages of type-based termination as well as comparing different approaches proposed in the literature (including explicit vs. implicit sizes, first-class sizes, and size inference). We will also discuss the challenges and implications of implementing sized types in the Coq kernel, both for the developers and users of Coq.

References

[1] Andreas Abel. *A Polymorphic Lambda-Calculus with Sized Higher-Order Types*. PhD thesis, Ludwig-Maximilians-Universität München, 2006.

[2] Bruno Barras. Sets in Coq, Coq in sets. *Journal of Formalized Reasoning*, 3(1):29–48, 2010.

[3] Gilles Barthe, Benjamin Grégoire, and Fernando Pastawski. CIC$\widehat{}$: Type-based termination of recursive definitions in the Calculus of Inductive Constructions. In Miki Hermann and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning, 13th International Conference, LPAR 2006, Phnom Penh, Cambodia, November 13-17, 2006, Proceedings*, volume 4246 of *Lecture Notes in Computer Science*, pages 257–271. Springer, 2006.

[4] Gilles Barthe, Benjamin Grégoire, and Colin Riba. Type-based termination with sized products. In Michael Kaminski and Simone Martini, editors, *Computer Science Logic, 22nd International Workshop, CSL 2008, 17th Annual Conference of the EACSL, Bertinoro, Italy, September 16-19, 2008. Proceedings*, volume 5213 of *Lecture Notes in Computer Science*, pages 493–507. Springer, 2008.

[5] Frédéric Blanqui. A type-based termination criterion for dependently-typed higher-order rewrite systems. In Vincent van Oostrom, editor, *Rewriting Techniques and Applications, 15th International Conference, RTA 2004, Aachen, Germany, June 3-5, 2004, Proceedings*, volume 3091 of *Lecture Notes in Computer Science*, pages 24–39. Springer, 2004.

[6] Pierre Boutillier. A relaxation of Coq's guard condition. In *Actes des Journées Francophones des langages Applicatifs*, pages 1–14, Carnac, France, February 2012.

[7] Jorge Luis Sacchini. *On Type-Based Termination and Dependent Pattern Matching in the Calculus of Inductive Constructions*. PhD thesis, École Nationale Supérieure des Mines de Paris, 2011.