Bootstrapping Fast Reflective Tactics for Reasoning Modulo AC with a Verified and Optimized Sorting Function in Coq (Progress Report)

Kazuhiko Sakaguchi*

University of Tsukuba, Japan

Abstract

This talk presents general building blocks for constructing reflective tactics for reasoning modulo associativity and commutativity (AC) in the Coq proof assistant [3]. *Proof by reflection* [1, Chapter 16] is a well-known method in type theory based formal proof systems for making proof construction/checking process much more efficient by relying on the convertibility test of terms. In the typical usage of reflection, repetitive and algorithmic rewriting process can be replaced by one rewriting, convertibility test, and computation of normal forms. Hereby size of proof terms can be drastically reduced.

This study improves the first step of reflective reasonings modulo AC, that is, normalization for terms freely generated by an AC binary operator $\star : S \to S \to S$ and terms of the type S. The normalization process consists of three steps: (1) indexing all atomic terms with totally ordered indices, (2) reifying (or quoting) the terms into an inductive data type representing binary trees, and (3) flattening and sorting the reified terms. Normalized terms always have the form of $f(i_1) \star \cdots \star f(i_n)$ where f is mapping function from indices to atomic terms, and the sequence of indices $i_1, \ldots, i_n$ are sorted by the given total order. Many automatic simplifications specialized for each commutative semigroup $(S, \star)$ and rewriting modulo AC can be performed efficiently by normalizing terms beforehand.

The fundamental idea of this study is an efficient construction technique of binary trees from lists which was originally provided by Georges Gonthier [2]. This method has many excellent properties for our usage: (I) the binary trees constructed by it are balanced, (II) it preserves the order of elements through the transformation from lists to trees, (III) all recursions used in it are structural, and (IV) the construction process of a tree from concatenated lists $X_1 +\!\!+ \cdots +\!\!+ X_n$ can be performed incrementally for each list X_i $(1 \leq i \leq n)$ without loss of efficiency.

Such the construction technique can be used for efficient implementations of the steps (1) and (3) — indexing atomic terms and sorting. For the step (1), we define a construction method of mapping function from binary representation indices to atomic terms by using the above construction technique. Thanks to the properties (I) and (II), we can obtain normalized terms such that the size of each index in them is bounded by $\mathcal{O}(\log n)$, and the order of first occurrences of each atomic term in input terms is preserved. For the step (2), we define a merge sort function by using the above construction technique. Thanks to the properties (II), (III), and (IV), this sorting function is stable, is defined without bounded or well-founded recursions, and can sort the reified terms directly without flattening into lists. Additionally, the sorting process in reflection can be done in $\mathcal{O}(n \log^2 n)$ time complexity thanks to the upper bound of indices size.

As an application of our reflective reasoning method, we implemented an `autoperm` tactic for equality up to permutation (i.e., multiset equality) of lists and a more efficient merge sort function with a tail-recursive merge. The permutation property of the new (partially tail-recursive) merge sort can be proved easily by using the `autoperm` tactic and can be used for reflective reasonings instead of older one. We will also report ongoing work which includes following two notable issues: is our indexing method useful to improve the efficiency of the `quote` tactic of Coq, and how can we improve the efficiency of reflective reasoning by using the new merge sort function?

References

[1] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions*. EATCS. Springer, 2004.

[2] Georges Gonthier. *RE: [Coq-Club] Sorting*. https://sympa.inria.fr/sympa/arc/coq-club/2009-04/msg00040.html. 2009.

[3] The Coq Development Team. *The Coq Proof Assistant Reference Manual*. Version 8.7.2. https://coq.inria.fr/distrib/V8.7.2/refman/. 2018.

*sakaguchi@coins.tsukuba.ac.jp