Automatically And Efficiently Illustrating Polynomial Equalities in Agda

Donnacha Oisín Kidney

December 9, 2018

Abstract

We present a new library which automates the construction of equivalence proofs between polynomials over commutative rings in the programming language Agda [12]. The library makes use of Agda's reflection machinery to provide an extremely simple interface, and is extremely flexible in its output, requiring only equivalence (not propositional equality) to construct proofs.

Contents

1 Introduction

Truly formal proofs of even basic mathematical identities are notoriously tedious and verbose. Perhaps the canonical example is Russell and Whitehead's proof that 1 + 1 = 2, which finally arrives on page 379 of Principia Mathematica [16].

More modern systems have greatly simplified the underlying formalisms, but they still often suffer from a degree of explicitness that makes elementary identities daunting. Dependently-typed programming languages like Agda [12] and Coq [14] are examples of such systems: used in the naïve way, equivalence proofs require the programmer to specify every individual step ("here we rely on the commutativity of +, followed by the associativity of × on its right side", and so on).

Coq and Agda are not just programming languages in name, though: they are fully-fledged and powerful, capable of producing useful software, including automated computer-algebra systems. Unlike most CASs, those written in Coq or Agda come with added guarantees of correctness in their operation. Furthermore, these systems can be used to automate the construction of identity proofs which would otherwise be too tedious to do by hand.

2 Related Work

The state-of-the-art solver for polynomial equalities (over commutative rings) was originally presented in [7], and is used in Coq's `ring` solver. This work improved on the already existing solver [5] in both efficiency and flexibility. In both the old and improved solvers, a reflexive technique is used to automate the construction of the proof obligation (as described in [1]).

Agda [12] is a dependently-typed programming language based on Martin-Löf's Intuitionistic Type Theory [9]. Its standard library [6] currently contains a ring solver which is similar in flexibility to Coq's `ring`, but doesn't support the reflection-based interface, and is less efficient due to its use of a dense (rather than sparse) internal data structure.

In [13], an implementation of an automated solver for the dependently-typed language Idris [2] is described. It uses type-safe reflection to provide a simple and elegant interface, and its internal solver algorithm uses a correct-by-construction approach. The solver is defined over *non*commutative rings, however, meaning that it is more general (can work with more types) but less powerful (meaning it can prove fewer identities). It does not use a sparse representation.

Reflection and metaprogramming are relatively recent additions to Agda, but form an important part of the interfaces to automated proof procedures. Reflection in dependent types in general is explored in [4], and specific to Agda in [15].

The progress of various formalization efforts is charted in [17]. DoCon [11] is a notable Agda library in this regard: its implementation and goal is described in [10]. [3] describes the manipulation of polynomials in both Haskell and Agda.

Finally, the study of *didactic* computer algebra systems is explored in [8].

3 Contributions

An New, Efficient Ring Solver We provide an implementation of a polynomial solver which uses the same optimizations described in [7] in the programming language Agda. Along the way, we demonstrate several techniques for writing efficient correct-by-construction code.

A Simple Reflection-Based Interface We use Agda's reflection machinery to provide the following interface to the solver:

```
lemma : ∀ x y →
  (x + y) ^ 2 ≈ x ^ 2 + y ^ 2 + 2 * x * y
lemma = solve NatRing
```

It imposes minimal overhead on the user: only the Ring implementation is required, with no need for user implementations of quoting. Despite this, it is generic over any type which implements ring.

A Didactic Computer-Algebra System As a result of the flexibility of the solver, the equivalence relation it constructs can be instantiated into a number of different forms (not just equality, for instance). While This has been exploited in Agda before to generate isomorphisms over containers, we use it here to construct didactic (or "step-by-step") solutions.

References

[1] S. Boutin, "Using reflection to build efficient and certified decision procedures," in *Theoretical Aspects of Computer Software*, ser. Lecture Notes in Computer Science, M. Abadi and T. Ito, Eds. Springer Berlin Heidelberg, 1997, pp. 515–529.

[2] E. Brady, "Idris, a general-purpose dependently typed programming language: Design and implementation," *Journal of Functional Programming*, vol. 23, no. 05, pp. 552–593, Sep. 2013. [Online]. Available: http://journals.cambridge.org/article_S095679681300018X

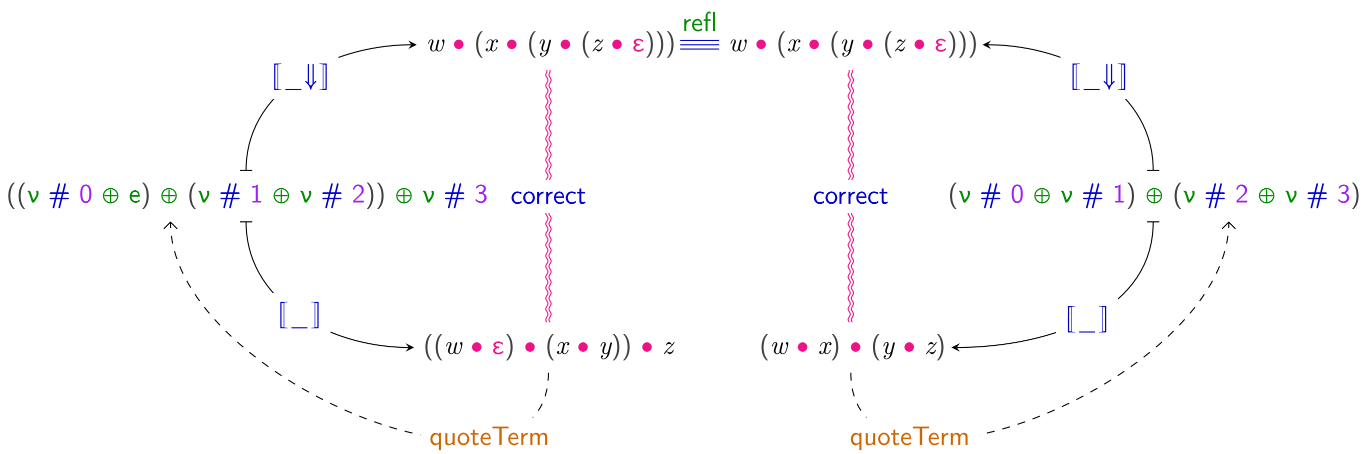
[3] C.-M. Cheng, R.-L. Hsu, and S.-C. Mu, "Functional Pearl: Folding Polynomials of Polynomials," in *Functional and Logic Programming*, ser. Lecture Notes in Computer Science. Springer, Cham, May 2018, pp. 68–83. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-90686-7_5

[4] D. R. Christiansen, "Practical Reflection and Metaprogramming for Dependent Types," Ph.D. dissertation, IT University of Copenhagen, Nov. 2015. [Online]. Available: http://davidchristiansen.dk/david-christiansen-phd.pdf

[5] T. Coq Development Team, *The Coq Proof Assistant Reference Manual, Version 7.2*, 2002. [Online]. Available: http://coq.inria.fr

[6] N. A. Danielsson, "The Agda standard library," Jun. 2018. [Online]. Available: https://agda.github.io/agda-stdlib/README.html

[7] B. Grégoire and A. Mahboubi, "Proving Equalities in a Commutative Ring Done Right



;

Figure 1: The Reflexive Proof Process

in Coq," in *Theorem Proving in Higher Order Logics*, ser. Lecture Notes in Computer Science, vol. 3603. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 98–113. [Online]. Available: http://link.springer.com/10.1007/11541868_7

[8] D. Lioubartsev, "Constructing a Computer Algebra System Capable of Generating Pedagogical Step-by-Step Solutions," Ph.D. dissertation, KTH Royal Institue of Technology, Stockholm, Sweden, 2016. [Online]. Available: http://www.diva-portal.se/smash/get/diva2:945222/FULLTEXT01.pdf

[9] P. Martin-Löf, *Intuitionistic Type Theory*, Padua, Jun. 1980. [Online]. Available: http://www.cse.chalmers.se/~peterd/papers/MartinL%00f6f1984.pdf

[10] S. D. Meshveliani, "Dependent Types for an Adequate Programming of Algebra," Program Systems Institute of Russian Academy of sciences, Pereslavl-Zalessky, Russia, Tech. Rep., 2013. [Online]. Available: http://ceur-ws.org/Vol-1010/paper-05.pdf

[11] ——, "DoCon-A a Provable Algebraic Domain Constructor," Pereslavl - Zalessky, Apr. 2018. [Online]. Available: http://www.botik.ru/pub/local/Mechveliani/docon-A/2.02/

[12] U. Norell and J. Chapman, "Dependently Typed Programming in Agda," p. 41, 2008.

[13] F. Slama and E. Brady, "Automatically Proving Equivalence by Type-Safe Reflection," in *Intelligent Computer Mathematics*, H. Geuvers, M. England, O. Hasan, F. Rabe, and O. Teschke, Eds. Cham: Springer International Publishing, 2017, vol. 10383, pp. 40–55. [Online]. Available: http://link.springer.com/10.1007/978-3-319-62075-6_4

[14] T. C. D. Team, "The Coq Proof Assistant, version 8.8.0," Apr. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1219885

[15] P. D. van der Walt, "Reflection in Agda," Master's Thesis, Universiteit of Utrecht, Oct. 2012. [Online]. Available: https://dspace.library.uu.nl/handle/1874/256628

[16] A. N. Whitehead and B. Russell, *Principia Mathematica. Vol. I*, 1910. [Online]. Available: https://zbmath.org/?q=an%3A41.0083.02

[17] F. Wiedijk, "Formalizing 100 Theorems," Oct. 2018. [Online]. Available: http://www.cs.ru.nl/~freek/100/