Reading and Writing Arithmetic: Automating Ring Equalities in Agda

ANONYMOUS AUTHOR(S)

We present a new library which automates the construction of equivalence proofs between polynomials over commutative rings and semirings in the programming language Agda [Norell and Chapman 2008]. It is asymptotically faster than Agda's existing solver. We use reflection to provide a simple interface to the solver, and demonstrate a novel use of the constructed relations: step-by-step solutions.

Additional Key Words and Phrases: proof automation, equivalence, proof by reflection, step-by-step solutions

```
lemma : ∀ x y → x + y * 1 + 3 ≈ 2 + 1 + y + x
```

```
lemma x y = begin
  x + y * 1 + 3 ≈⟨ refl ⟨ +-cong ⟩ *-identityʳ y ⟨ +-cong ⟩ refl {3} ⟩
  x + y + 3     ≈⟨ +-comm x y ⟨ +-cong ⟩ refl ⟩
  y + x + 3     ≈⟨ +-comm (y + x) 3 ⟩
  3 + (y + x)   ≈⟨ sym (+-assoc 3 y x) ⟩
  2 + 1 + y + x ∎
```

(a) A Tedious Proof

```
lemma = solve NatRing
```

(b) The Solver

Fig. 1. Comparison Between A Manual Proof and The Automated Solver

1 INTRODUCTION

Doing mathematics in dependently-typed programming languages like Agda has a reputation for being tedious, awkward, and difficult. Even simple arithmetic identities like the one in Fig. 1 require fussy proofs (Fig. 1a).

This need not be the case! With some carefully-designed tools, mathematics in Agda can be easy, friendly, and fun. This work describes one such tool: an Agda library which automates the construction of these kinds of proofs, making them as easy as Fig. 1b.

As you might expect, our solver comes accompanied by a formal proof of correctness. Beyond that, though, we also strove to satisfy the following requirements:

Friendliness and Ease of Use Proofs like the one in Fig. 1a aren't just boring: they're *difficult*. The programmer needs to remember the particular syntax for each step ("is it +-comm or +-commutative?"), and often they have to put up with poor error messages.

We believe this kind of difficulty is why Agda's current ring solver [Danielsson 2018] enjoys little widespread use. Its interface (Fig. 2) is almost as verbose as the manual proof, and it requires programmers to learn another syntax specific to the solver.

Our solver strives to be as easy to use as possible: the high-level interface is simple (Fig. 1b), we don't require anything of the user other than an implementation of one of the supported algebras, and effort is made to generate useful error messages.

2018. 2475-1421/2018/1-ART1 $15.00
https://doi.org/

```
lemma = +-*-Solver.solve 2 (λ x y → x :+ y :* con 1 :+ con 3 := con 2 :+ con 1 :+ y :+ x) refl
```

Fig. 2. The Old Solver

Performance Typechecking dependently-typed code is a costly task. Automated solvers like the one presented here can greatly exacerbate this cost: in our experience, it wasn't uncommon for Agda's current ring solver to spend upwards of 10 minutes proving a single expression. In practice, this means two things: firstly, formalizations of large sections of mathematics can potentially take hours to typecheck (by which time the programmer has understandably given up on mathematics altogether); secondly, certain identities can take simply too long to prove, effectively making them unprovable.

There are well-known techniques for optimizing the kind of solver we provide here, most of them described in [Grégoire and Mahboubi 2005]. Translating these optimizations into Agda was far from straightforward, however, and we found the performance benefits to be quite fragile. We did eventually achieve consistent and significant speedups, though, with very different techniques.

Educational Features While our solver comes with the benefit of formal correctness, it's still playing catch-up to other less-rigorous computer algebra systems in terms of features. These features have driven systems like Wolfram|Alpha [Wolfram Research, Inc. 2019] to widespread popularity among (for instance) students learning mathematics.

We will take just one of those features ("pedagogical", or step-by-step solutions [The Development Team 2009]), and reimplement it in Agda using our solver. In doing so, we will explore some of the theory behind it, and present a formalism to describe these solutions.

REFERENCES

Nils Anders Danielsson. 2018. The Agda Standard Library. https://agda.github.io/agda-stdlib/README.html

Benjamin Grégoire and Assia Mahboubi. 2005. Proving Equalities in a Commutative Ring Done Right in Coq. In *Theorem Proving in Higher Order Logics (Lecture Notes in Computer Science)*, Vol. 3603. Springer Berlin Heidelberg, Berlin, Heidelberg, 98–113. https://doi.org/10.1007/11541868_7

Ulf Norell and James Chapman. 2008. Dependently Typed Programming in Agda. (2008), 41.

The Development Team. 2009. Step-by-Step Math. http://blog.wolframalpha.com/2009/12/01/step-by-step-math/

Wolfram Research, Inc. 2019. Wolfram|Alpha. Wolfram Research, Inc.. https://www.wolframalpha.com/

A APPENDIX

Text of appendix . . .