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 a dependently-typed programming languages like Agda [Norell and Chapman 2008] 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, to deal with equalities over commutative rings and semirings.

1.1 Contributions

We present a library which automates the solving of equalities over commutative rings and semirings in Agda. In writing this library, we had three main goals:

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.
Even though Agda's standard library [Danielsson 2018] currently has a ring solver, its interface is almost as verbose as the manual proof, and it requires users write the goal twice, once in the signature and again in the specific syntax used by the solver (Fig. 2).

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

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.

Efficiency Typechecking dependently-typed code is a costly task. Automated solvers, like the one presented here, can seriously add to that cost, occasionally to the extent that some identities are simply infeasible to prove.

Polynomials are represented internally in sparse Horner normal form. Manipulation of this internal representation uses many of the same optimizations as in [Grégoire and Mahboubi 2005], although their implementation proved to be quite difficult in Agda. Furthermore, we found that the real performance bottleneck lied elsewhere, so overall our strategy for optimization was quite different. The end result is that our solver is asymptotically faster than Agda's current.

Educational

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.

A APPENDIX

Text of appendix ...