Automatically and Efficiently Illustrating Polynomial Equalities in Agda

Donnacha Oisín Kidney

March 26, 2019

Agda

Agda is both a mathematical formalism and an executable programming language.

As a programming language, it is similar to Haskell.

```
reverse : ∀ {a} {A : Set a} → List A → List A
reverse [] = []
reverse (x :: xs) = reverse xs ++ x :: []
```

Ulf Norell and James Chapman. Dependently Typed Programming in Agda.

2008

Mathematical Formalisms

Classical
Zermelo-Fraenkel Set Theory

Constructivist

Martin-Löf's Intuitionistic Type Theory	Agda, Idris
Calculus of Constructions	Coq

Axiom of Choice

$$\forall P . P \vee \neg P$$

Proof by Contradiction

$$\forall P . \neg\neg P \to P$$

Two Problems

Formalisms are too Verbose

A. N. Whitehead and B. Russell. *Principia Mathematica. Vol. I.* **1910**

$1 + 1 = 2$ proven on page 360.

Automation is Untrustworthy

Kenneth Appel and Wolfgang Haken. The Solution of the Four-Color-Map Problem. ***Scientific American*, 237(4):108–121, 1977**

Researchers began to notice that the type systems of programming languages might be a good solution for both of these problems.

By the Curry-Howard isomorphism, proofs are programs, and programs are proofs.

This Project

$$\text{lemma} : \forall\ x\ y \to x + y * 1 + 3 \approx 2 + 1 + y + x$$

```
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 ∎
```

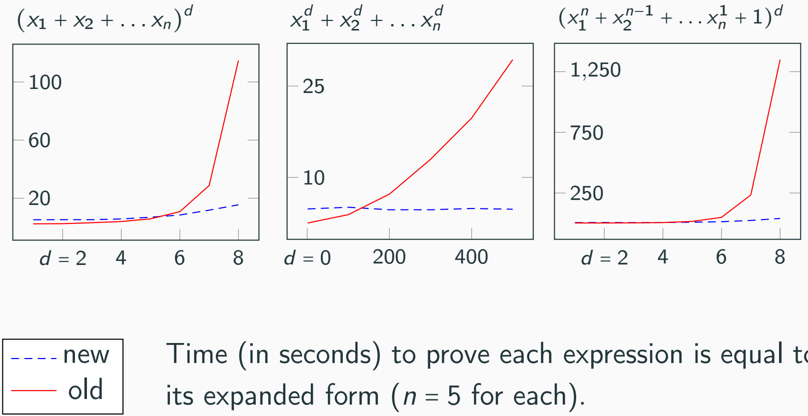
Figure 1: A Tedious Proof

```
lemma = solve NatRing
```

Figure 2: Our Solver

The Algorithm

We convert to Horner Normal Form, and prove the conversion correct.



Time (in seconds) to prove each expression is equal to its expanded form ($n = 5$ for each).

Didactic Solutions

Another implication of "Proofs are programs" is that proofs have computational content.

For instance, the "proof" of equality can be a path. These form equivalence relations, where equivalence classes are connected components in the graph.

We can then perform some cleaning-up of the path (an A*-like algorithm, as well as some heuristics), and print the result out to the user.

```
x + y + 3
    ={ +-comm(x, y + 3) }
y + 3 + x
    ={ +-comm(y, 3) }
3 + y + x
```

Questions?