Programming Mathematics in Agda

Donnacha Oisín Kidney

October 13, 2018

What do Programming Languages Have to Say About Mathematics?

Programming is Proving

A Polynomial Solver

The p-Adics

What do Programming Languages Have to Say About Mathematics?

Languages for proofs and languages for programs have a lot of the same requirements.

Languages for proofs and languages for programs have a lot of the same requirements.

A *Syntax* that is

- Readable
- Precise
- Terse

Languages for proofs and languages for programs have a lot of the same requirements.

A *Syntax* that is

- Readable
- Precise
- Terse

***Semantics* that are**

- Small
- Powerful
- Consistent

Why not use a programming language as our proof language?

Benefits For Programmers

Benefits For Programmers

- *Prove* things about code

```
assert(list(reversed([1,2,3])) == [3,2,1])
```

vs

$$\text{reverse-involution} : \forall\ xs \to \text{reverse}\ (\text{reverse}\ xs) \equiv xs$$

Benefits For Programmers

- *Prove* things about code
- Use ideas and concepts from maths—why reinvent them?

Benefits For Programmers

- *Prove* things about code
- Use ideas and concepts from maths—why reinvent them?
- Provide coherent *justification* for language features

Benefits For Mathematicians

Benefits For Mathematicians

- Have a machine check your proofs

Currently, though, this is *tedious*

Benefits For Mathematicians

- Have a machine check your proofs
- Run your proofs

Benefits For Mathematicians

- Have a machine check your proofs
- Run your proofs
- Develop a consistent foundation for maths

Benefits For Mathematicians

- Have a machine check your proofs
- Run your proofs
- Develop a consistent foundation for maths

Wait—isn't this impossible?

Formalizing Mathematics

Lawrence C Paulson. The Future of Formalised Mathematics, 2016

Formalizing Mathematics

Whitehead and Russell took
hundreds of pages to prove
$1 + 1 = 2$

Lawrence C Paulson. The Future of Formalised Mathematics, 2016

Formalizing Mathematics

Whitehead and Russell took *hundreds* of pages to prove $1 + 1 = 2$

Gödel showed that universal formal systems are incomplete

Lawrence C Paulson. The Future of Formalised Mathematics, 2016

Formalizing Mathematics

Whitehead and Russell took *hundreds* of pages to prove $1 + 1 = 2$

Formal systems have improved

Gödel showed that universal formal systems are incomplete

Lawrence C Paulson. The Future of Formalised Mathematics, 2016

Formalizing Mathematics

Whitehead and Russell took *hundreds* of pages to prove $1 + 1 = 2$

Formal systems have improved

Gödel showed that universal formal systems are incomplete

We don't need universal systems

Lawrence C Paulson. The Future of Formalised Mathematics, 2016

What About Automated Theorem Provers?

What About Automated Theorem Provers?

Appel and Haken's 1976 proof of the Four-Color Theorem

What About Automated Theorem Provers?

Appel and Haken's 1976 proof of the Four-Color Theorem

This is about picking a set of axioms and syntax so simple that *even a computer* could understand them.

What About Automated Theorem Provers?

Appel and Haken's 1976 proof of the Four-Color Theorem

This is about picking a set of axioms and syntax so simple that *even a computer* could understand them.

Formalized proof of the Four-Colour theorem arrived a full *29 years* later!

Georges Gonthier. Formal Proof—The Four-Color Theorem.

***Notices of the AMS*, 55(11):12, 2008**

What About Automated Theorem Provers?

Appel and Haken's 1976 proof of the Four-Color Theorem

This is about picking a set of axioms and syntax so simple that *even a computer* could understand them.

Formalized proof of the Four-Colour theorem arrived a full *29 years* later!

Georges Gonthier. Formal Proof—The Four-Color Theorem.

***Notices of the AMS*, 55(11):12, 2008**

Since our proof language is *executable*, maybe we can write *verified* automated theorem provers?

Programming is Proving

The Curry-Howard Correspondence

$$
\begin{array}{ccc}
\textit{Type} & \Longleftrightarrow & \textit{Proposition} \\
\downarrow & & \downarrow \\
\textit{Program} & \Longleftrightarrow & \textit{Proof}
\end{array}
$$

Philip Wadler. Propositions As Types.

***Commun. ACM*, 58(12):75–84, November 2015**

Types are Propositions

Types are (usually):

- Int
- String
- ...

How are these propositions?

Existential Proofs

Existential Proofs

So when you see:

$$x : \mathbb{N}$$

Existential Proofs

So when you see:	Think:
$x : \mathbb{N}$	$\exists . \mathbb{N}$

Existential Proofs

So when you see:	Think:
$x : \mathbb{N}$	$\exists.\mathbb{N}$

NB
We'll see a more powerful and precise version of $\exists$ later.

Existential Proofs

So when you see: Think:

$x : \mathbb{N}$ $\exists . \mathbb{N}$

NB
We'll see a more powerful and precise version of $\exists$ later.

Proof is "by example":

So when you see:	Think:
$x : \mathbb{N}$	$\exists.\mathbb{N}$

NB
We'll see a more powerful and precise version of $\exists$ later.

Proof is "by example":

$x = 1$

Programs are Proofs

Programs are Proofs

```
>>> head [1,2,3]
1
```

```
>>> head [1,2,3]
1
```

Here's the type:

$$\mathsf{head} : \{A : \mathsf{Set}\} \rightarrow \mathsf{List}\ A \rightarrow A$$

Basic Agda Syntax

Basic Agda Syntax

Basic Agda Syntax

Equivalent in other languages:

Haskell	`head :: [a] -> a`
Swift	`func head<A>(xs : [A]) -> A {`

Basic Agda Syntax

Equivalent in other languages:

Haskell	`head :: [a] -> a`
Swift	`func head<A>(xs : [A]) -> A {`

$\mathsf{head} : \{A : \mathsf{Set}\} \to \mathsf{List}\ A \to A$

Basic Agda Syntax

Equivalent in other languages:

Haskell	`head :: [a] -> a`
Swift	`func head<A>(xs : [A]) -> A {`

$\text{head} : \{A : \text{Set}\} \to \text{List}\ A \to A$ “Takes a list of things, and returns one of those things”.

The Proposition is False!

```
>>> head []
error "head: empty list"
```

The Proposition is False!

```
>>> head []
error "head: empty list"
```

$\mathsf{head} : \{A : \mathsf{Set}\} \to \mathsf{List}\ A \to A$

The Proposition is False!

```
>>> head []
error "head: empty list"
```

$\mathsf{head} : \{A : \mathsf{Set}\} \to \mathsf{List}\ A \to A$

False

If Agda is correct (as a formal logic):

If Agda is correct (as a formal logic):

We shouldn't be able to prove this using Agda

If Agda is correct (as a formal logic):

We shouldn't be able write this function in Agda

But Let's Try Anyway! i

Function definition syntax

$$
\begin{aligned}
&\mathsf{fib} : \mathbb{N} \to \mathbb{N} \\
&\mathsf{fib}\ 0 && = 0 \\
&\mathsf{fib}\ (1\!+\ 0) && = 1\!+\ 0 \\
&\mathsf{fib}\ (1\!+\ (1\!+\ n)) && = \mathsf{fib}\ (1\!+\ n) + \mathsf{fib}\ n
\end{aligned}
$$

But Let's Try Anyway! ii

$$
\begin{aligned}
&\mathsf{length} : \{A : \mathsf{Set}\} \to \mathsf{List}\ A \to \mathbb{N} \\
&\mathsf{length}\ [] = 0 \\
&\mathsf{length}\ (x :: xs) = 1 + \mathsf{length}\ xs
\end{aligned}
$$

But Let's Try Anyway! iii

Here's a definition for head:

$$\text{head } (x :: xs) = x$$

No!
For correct proofs, partial functions aren't allowed

But Let's Try Anyway! iv

We're not out of the woods yet:

$$\text{head } [] = \text{head } []$$

No!
For correct proofs, all functions must be total

Correctness

For the proofs to be correct, we have two extra conditions that you usually don't have in programming:

- No partial programs
- Only total programs

Can we *prove* that head doesn't exist?

Falsehood

Falsehood

Principle of Explosion
"Ex falso quodlibet"

If you stand for nothing, you'll fall for anything.

Falsehood

$\neg : \forall \{\ell\} \to \mathsf{Set}\ \ell \to \mathsf{Set}\ \_$
$\neg\ A = A \to \{B : \mathsf{Set}\} \to B$

Principle of Explosion
"Ex falso quodlibet"

If you stand for nothing, you'll fall for anything.

$$\mathsf{head\text{-}doesn't\text{-}exist} : \neg\,(\{A : \mathsf{Set}\} \to \mathsf{List}\ A \to A)$$
$$\mathsf{head\text{-}doesn't\text{-}exist}\ \mathit{head} = \mathit{head}\ []$$

A Polynomial Solver

The p-Adics