A *way too brief* introduction to Agda

Daniel S. Fava

University of Oslo, Norway

March, 2019

Recapitulation

Curry-Howard

Correspondence between *logics* and *models of computation*

- natural deduction ⇔ lambda calculus
- Hilbert-style deduction ⇔ combinators

Example:

`f : A→B`

`f` as a function from a proof `a` of proposition `A` to a proof `f(a)` of `B` [Brouwer-Heyting-Kolmogorov]

- Types as logical formulas, programs as proofs.
- “Type `A` is inhabited” as “proposition `A` has a proof.”

Recapitulation

False formula (logic) ⇔ bottom type (computation)

Constructive: $\neg A$ means $A \to \bot$

- $\neg\neg A$ does not mean A
- $\neg\neg A$ means $\neg(A \to \bot)$ which means $(A \to \bot) \to \bot$

Want a language where program, specification, and proofs are all under the same umbrella.

Dependent types

Rich types that more closely express program behavior.

Type inference is so last century.

The right thing to do is to write the types down, and then get as much mechanical assistance generating the program as possible.

Conor McBride

Internal vs. External verification

Example: Creates a list of `n` elements `a : A`

```
gen 3 "uio" ≡ ["uio", "uio", "uio"]
```

- Using `List A`
  - No internal verification
  - Must proof two invariants externally: size and constant element
- Using `Vec A (n : ℕ)`
  - Internal verification: Size is part of `Vec`'s type
  - Must prove constant element externally
- Define `UVec (a : A) (n : ℕ)`
  - Both properties (size and constant element) are internally verified

Pi type

In addition to the function type `A→B`, we have the Pi type:

`Π(x:A) → B(x)`

where `B(x)` is a *type family*

Pi type (programming) ⇔ Universal quantification (logic)

Equality type

`a ≡ b`

- Normalize (apply definitions) both sides then compare.
- Equality type is proven with `refl`, in other words,
  - `refl` is a constructor for the equality type.

Notions of equality

- definitional (intensional)
- propositional (extensional)
- computational

Type erasure

- A vector is just a list after type erasure
  - Invariant related to size is checked statically
- What can be erased and what can't?
  - At the crux of making the language efficient