Better Software with Better Types

Wouter Swierstra
University of Nottingham

Foundations of Programming
University of Nottingham

Areas of Expertise

- Category Theory
- Type Theory
- Functional Programming

Type Systems

- Strong type systems catch a lot of bugs before a program is ever run...
- .. but things can still fail dynamically!
- For instance:

```
public int evil(int x)
  { return x/0; }
```

Can we do better?

Functional Programming

- Haskell programmers claim "if a program compiles it is correct".

```
data List = Nil
  | Cons Int List
```

- The `tail` function is partial:

```
tail :: List -> List
tail (Cons x xs) = xs
tail Nil         = ??
```

The Challenge

- Types give approximate information.
- Sometimes we want to be more precise:

```
tail :: List_{n+1} -> List_n
```

- This pops up all the time: array indexing, division by zero, ...
- But conventional type system don't provide this kind of guarantee.

```
------------------------------------------
      (  xs  :  Vec  (suc n)  A   !
let   !-------------------------!
      !  vtail xs  :  Vec  (n)  A  )

      vtail xs <= case xs
      { vtail (vcons n' ns) => ns
      }
------------------------------------------
```

Epigram

Simple types

$$\frac{\mathtt{x} + \mathtt{1} : \mathtt{Int} \qquad \text{using } \mathtt{x} : \text{Int}}{\lambda \mathrm{x}.\mathrm{x} + \mathtt{1} : \text{Int} \to \text{Int}}$$

$$\frac{\mathtt{isZero} : \mathtt{Int} \to \mathtt{Bool} \qquad \mathtt{5} : \mathtt{Int}}{\mathtt{isZero}(\mathtt{5}) : \mathtt{Bool}}$$

How much information can these types give us?

Simple Types

$$\frac{\Gamma, x : S \vdash t : T}{\Gamma \vdash \lambda x.t : S \to T}$$

$$\frac{\Gamma \vdash f : S \to T \qquad \Gamma \vdash s : S}{\Gamma \vdash f(s) : T}$$

Propositional Calculus

Type systems:

$$\frac{\Gamma \vdash f : S \to T \qquad \Gamma \vdash s : S}{\Gamma \vdash f(s) : T} \qquad \frac{\Gamma, x : S \vdash t : T}{\Gamma \vdash \lambda x.t : S \to T}$$

Logic:

$$\frac{\Gamma \vdash s \Rightarrow t \qquad \Gamma \vdash s}{\Gamma \vdash t} \qquad \frac{\Gamma, s \vdash t}{\Gamma \vdash s \Rightarrow t}$$

The Curry-Howard Isomorphism

- Types are propositions.
- Proofs are programs.
- Propositional calculus is pretty weak...
- What about predicate calculus?

Dependent Types

What is the type of `printf`?

```
printf("%s is %d","Wouter",24)
```

The number of arguments depends on the formatting string!

Dependent Types

$$\frac{\Gamma, x : S \vdash t : T}{\Gamma \vdash \lambda x.t : (x : S) \to T}$$

$$\frac{\Gamma \vdash f : (x : S) \to T \qquad \Gamma \vdash s : S}{\Gamma \vdash f(s) : T[x/s]}$$

Alternative notations:

$$(x : S) \to T \qquad \Pi x : S.T \qquad \forall x : S.T$$

Enter Dependent types

- Types express properties of values.
- To express meaningful properties, we need values to appear within types.

```
tail :: List_{n+1} -> List_n
redBlackTree :: RBTree Black 4
```

- We can capture arbitrary properties and invariants of our programs in their type!
- Yet the type system fits on a beer coaster.

Dependent types

- Dependent types were introduced by Per Martin Löf to formalise mathematics.
- We can prove every constructively valid formula in predicate logic.
- Dependent types form the basis of lots of theorem provers (Coq, Agda, ...)
- Proving a theorem is writing a term.

Dependent Types in Action

- Most research is biased towards theorem proving.
- Recently, interest has shifted towards programming with dependent types.

Epigram

- A purely functional language with dependent types.
- Interactive programming environment.
- Types guarantee program behaviour.
- Pay-as-you go program correctness.
- You can reason about programs you write without an external tool.

Demo

Not Everything in the Garden is Rosy.

Farewell Phase distinction

- How should we unify `Vec 3 Int` and `Vec (1+2) Int`?
- Unification must evaluate terms.
- What about statically evaluating `T (..formatHardDisk..)`?
- Can type checking diverge?

All our functions must be pure and total.

Purity and Totality

- Total functions are guaranteed to return a result for every input.
- Pure functions are \`perfectly predictable'
  - have no side-effects – that means no destructive updates, interaction with users, random numbers, ...
  - the result only depends on the input and is completely context insensitive.

Recent Work

- What is a pure and total webserver?
- We want real-world nasties into our beautiful language.
- Construct a faithful pure model.
- Then we can permit: teletype I/O, mutable state, concurrency, non-termination, ...

What are Integers?

- We cannot expect programmers to write `suc(suc(suc zero)` instead of `3`.
- We need better support for programming with integers.
- Programmers use different integers all the time (counters, divide-and-conquer, modular arithmetic).

Bigger Design Space

- Do you want lists or vectors? Or both?
- We need good support for generic programming.
- We need to find the right libraries.
- We need to facilitate code refactoring.

Compiler technology

- How should we compile dependently typed languages?
- Types don’t slow us down – they give us the opportunity to optimize.
- Can we provide some kind of type inference?

References

- Epigram:

  Conor McBride
  Epigram: Practical Programming with Dependent Types
  Summer School on Advanced Functional Programming, 2004

- Curry-Howard:

  Phil Wadler, *New Languages, Old Logic*, Dr. Dobbs Journal

- Why Dependent Types Matter

  James McKinna – POPL 2006