The Quite Reasonable Effectiveness of Formalization in Programming Language Design

Jamie Fulford
Advisors: Steve Zdancewic, Stephen Mell

Department of Computer and Information Science
The University of Pennsylvania

REPL, August 2, 2024

"Quite Reasonable Effectiveness"

"Quite Reasonable Effectiveness"

"Quite Reasonable Effectiveness"

- Eugene Wigner's "Unreasonable Effectiveness of Mathematics"

"Quite Reasonable Effectiveness"

- Eugene Wigner's "Unreasonable Effectiveness of Mathematics"
- What is reasonable effectiveness?

The Experience of Formalization

The Experience of Formalization

- Expection: Proofs are hard.

The Experience of Formalization

- Expection: Proofs are hard.
- Reality: Proofs are easy, *models* are hard.

The Experience of Formalization

- Expection: Proofs are hard.
- Reality: Proofs are easy, *models* are hard.
- "Premature optimization is the root of all evil."

The Experience of Formalization

- Expection: Proofs are hard.
- Reality: Proofs are easy, *models* are hard.
- "Premature *formalization* is the root of all evil."

The Experience of Formalization

- Expection: Proofs are hard.
- Reality: Proofs are easy, *models* are hard.
- "Premature *formalization* is the root of all evil."
- "Premature *abstraction* is the root of all evil."

Premature Abstraction

```
(* We define our total category of graphs
   which is fibered over the base category *)
Section TotalCategoryG.
  Context `{Countable K}.
  Context `{FMap SH}.
  Context {elts : forall A, Elements A (SH A)}.

  Definition G_obj :=
  { G : graph | @GraphWF K _ _ SH _ G }.

  Definition G_hom (A B : G_obj) :=
    { f : B_hom (interface (`A)) (interface (`B)) &
      forall n, ElemOf n dom (node (`A)) ->
        exists m, ElemOf m dom (node (`B)) /\
  ...
```

The Root of All Evil

Abstraction as a luxury

- Premature optimization
- Moves away from the problem

The Root of All Evil

Abstraction as a luxury

- Premature optimization
- Moves away from the problem

Abstraction as a necessity

- Occam's Razor
- Minimally sufficient model

Case Study: Abstraction as a Necessity

The EPIC language

- Confluent, nondeterministic lambda calculus variant
- Parallel by default
- Tree-like denotation

Case Study: Abstraction as a Necessity

The EPIC language

- Confluent, nondeterministic lambda calculus variant
- Parallel by default
- Tree-like denotation

```
Inductive term :=
| lam : lets -> term
with lets :=
| def (t:term) (l:lets)
| app (f:id) (x:id) (l:lets)
| tpl (xs:list id) (l:lets)
| prj (n:nat) (x:id) (l:lets)
| cut (x:id) (l:lets)
| ret (y:id).
```

Case Study: Abstraction as a Necessity

Definition

A term is **well-formed** if each of its `lets` fits within the "context size" (the amount of `lets`).

Case Study: Abstraction as a Necessity

Definition

A term is **well-formed** if each of its `lets` fits within the "context size" (the amount of `lets`).

Problem

How do we include the notion of alpha variance into our well-formedness function?

Case Study: Abstraction as a Necessity

Definition

A term is **well-formed** if each of its `lets` fits within the "context size" (the amount of `lets`).

Problem

How do we include the notion of alpha variance into our well-formedness function?

Solution

Define `id := nat`.

Case Study: Abstraction as a Necessity

Definition

A term is **well-formed** if each of its `lets` fits within the "context size" (the amount of `lets`).

Problem

How do we include the notion of alpha variance into our well-formedness function?

Solution

Define `id := nat.` (De Bruijn indeces).

Case Study: Abstraction as a Necessity

Definition

A term is **well-formed** if each of its `lets` fits within the "context size" (the amount of `lets`).

Problem

How do we include the notion of alpha variance into our well-formedness function?

Solution

Define `id := nat.` (De Bruijn indeces).

```
Definition wf_var (G : nat) (x : id) : bool := x <? G.
```

Case Study: Abstraction as a Luxury

Unfortunately...

Case Study: Abstraction as a Luxury

Unfortunately... most of Category Theory

The Quite Reasonable Benefits of Formalization

- Reveals assumptions, resolves inconsistencies, rigorous.

The Quite Reasonable Benefits of Formalization

- Reveals assumptions, resolves inconsistencies, rigorous.
- Performed by a human, verified by a computer

The Quite Reasonable Benefits of Formalization

- Reveals assumptions, resolves inconsistencies, rigorous.
- Performed by a human, verified by a computer
- Programming languages allow us to define an *effective procedure* for a computer.

The Quite Reasonable Benefits of Formalization

- Reveals assumptions, resolves inconsistencies, rigorous.
- Performed by a human, verified by a computer
- Programming languages allow us to define an *effective procedure* for a computer.
- Formalization is the *effective procedure* of mathematics.

Why?

Why?

Why Formalize?

Well...

Why?

Why Formalize?

Well... isn't formalization *exactly* what mathematics is?

Why?

Why Formalize?

Well... isn't formalization *exactly* what mathematics is?

Why Mathematics?

Well...

Why?

Why Formalize?

Well... isn't formalization *exactly* what mathematics is?

Why Mathematics?

Well... because it's *unreasonably* effective.

Why?

Why Formalize?

Well... isn't formalization *exactly* what mathematics is?

Why Mathematics?

Well... because it's *unreasonably* effective.

□