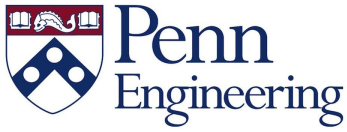
Verification of System FC in Coq

Tiernan Garsys, Tayler Mandel, Lucas Peña, Noam Zilberstein



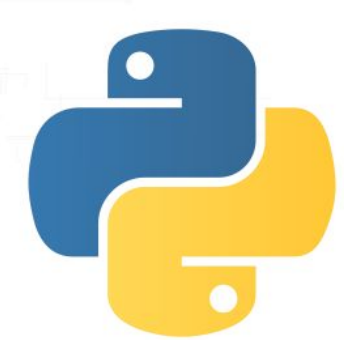
Advised by Stephanie Weirich & Richard Eisenberg

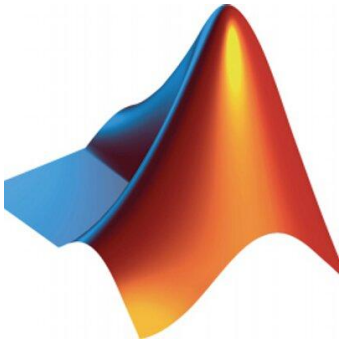
Introduction

- Haskell
  - Statically Typed
  - Functional
  - Pure
- Used for trusted type system
- Key question: Is Haskell really type safe?

Introduction

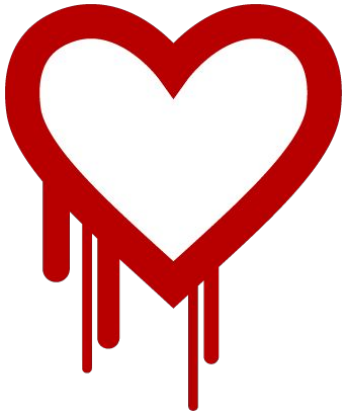
- Static Typing
  - Types are checked at *compile time*
  - Provides guarantees about program behavior
- Alternative: Dynamic Typing
  - Type errors at *runtime*
  - Ex: Python, MATLAB





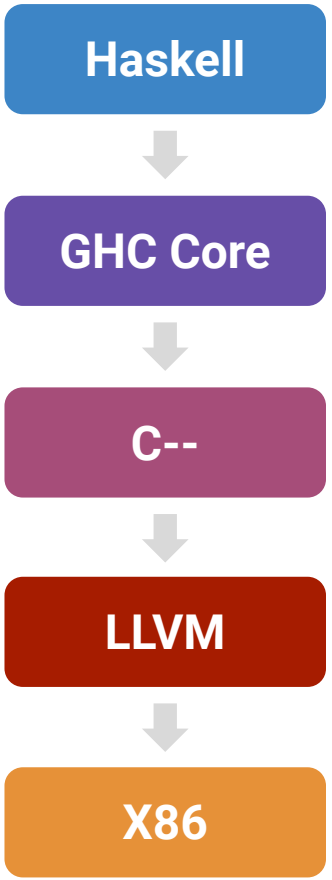
Project Motivation

- Software controls safety critical systems
  - Flight controllers
  - Self-driving cars
- Unsafe languages have severe consequences
  - Heartbleed
  - Shellshock



Haskell Compilation Process

- Frontend: Haskell to GHC Core
  - System FC is the formalization of GHC Core
  - Optimization passes at GHC Core level
- Type system of System FC is equivalent to that of Haskell

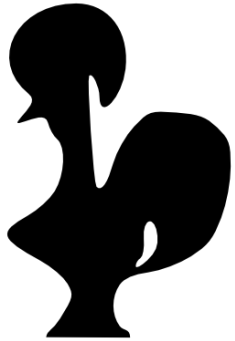


Type Safety

- Types dictate how expressions can be manipulated
- Prevent program from reaching inconsistent states
- Haskell's type system has not been mechanically proven to be sound

Formal Verification

- Curry-Howard Isomorphism
  - Types ~ Theorems
  - Programs ~ Proofs
- Coq
  - Dependently Typed, Functional Language
  - Automated Proof Assistant
- Approach
  - Formalize semantics of System FC in Coq
  - Prove type safety theorems



Key Theorems

- Progress
  - Any well typed expression can either take a step, or is a value
  - Well-typed expressions are not *stuck*

```
Theorem progress : forall t T,
     empty |- t \in T ->
     value t \/ exists t', t ==> t'.
```

Key Theorems

- Preservation
  - Evaluating an expression cannot change its type

```
Theorem preservation : forall t t' T,
    Gamma |- t \in T ->
    t ==> t'         ->
    Gamma |- t' \in T.
```

Key Theorems

- Soundness
  - Corollary of Progress and Preservation
  - Any well-typed expression evaluates to a consistent state

```
Corollary soundness : forall t t' T,
    empty |- t \in T ->
    t ==>* t'        ->
    ~(stuck t').
```

Results

- Formalization of System FC
  - Functions
  - Polymorphic types
  - Coercions
- Verified Proofs
  - Progress
  - Preservation
  - Soundness

$$\Lambda T.\, \lambda x : T.\, x$$

Next Steps

- Additional features of System FC
  - Data types
  - Type families (type-level functions)
- Verify translation to GHC Core
- Prove compiler optimizations
  - Code transformations should preserve semantics
- Prove language extensions

Questions?