

# Refinement Types for Elm

Second Master Thesis Report

---

Lucas Payr

31 January 2019

# Current State

1. Formal language similar to Elm
  - 1.1 A formal syntax **(DONE)**
  - 1.2 A formal type system **(DONE)**
  - 1.3 A denotational semantic **(DONE)**
  - 1.4 A small step semantic (using K Framework)  
**(WORK IN PROGRESS)**
  - 1.5 Proof that the type system is valid with respect to the semantics.
2. Extension of the formal language with Liquid Types
3. A type checker implementation written in Elm for Elm.

# Topics of this Talk

- Quick introduction to K Framework
- Discuss the formal type system based on an example
- Compare the implemented type checker in K Framework with the formal type system
- Live demonstration using the given example.

- Created in 2003 by Grigore Rosu
- Maintained and developed by the research groups FSL (Illinois,USA) and FMSE (Lasi,Romania).
- Framework for designing and formalizing programming languages.
- Based on Rewriting systems.

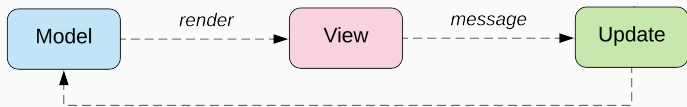
## Background: Elm Programming Language

- Invented by Evan Czaplicki as his master-thesis in 2012.
- Goal: Bring Function Programming to Web-Development
- Side-Goal: Learning-friendly design decisions
- Website: [elm-lang.org](http://elm-lang.org)

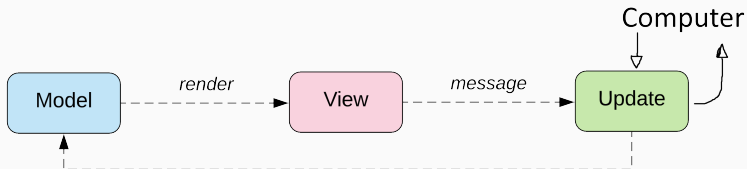
### Characteristics

- Pure Functional Language (immutable, no side effect, everything is a function)
- Compiles to JavaScript (in the future also to WebAssembly)
- ML-like Syntax (we say `fun a b c` for *fun(a, b, c)*)
- Simpler than Haskell (no Type classes, no Monads, only one way to do a given thing)
- “No Runtimes errors” (running out of memory, function equality and non-terminating functions still give runtime errors.)

## Background: The Elm Architecture



# Background: The Elm Architecture



## Example

- Online Editor: [ellie-app.com](https://ellie-app.com)

## Background: Refinement Types

Restricts the values of an existing type using a predicate.

Initial paper in 1991 by Tim Freeman and Frank Pfenning

- Initial concept was done in ML.
- Allows predicates with only  $\wedge, \vee, =$ , constants and basic pattern matching.
- Operates over algebraic types.
- Needed to specify **explicitly** all possible Values.

### Example

$$\{a : (Bool, Bool) \mid a = (True, False) \vee a = (False, True)\}$$
$$\forall t. \{a : List\ t \mid a = Cons\ (b : t)\ (c : List\ t) \wedge c = Cons\ (d : t)\ []\}$$



## Background: Liquid Types

Liquid Types (Logically Quantified Data Types) introduced in 2008

- Invented by Patrick Rondon, Ming Kawaguchi and Ranji Jhala
- Initial concept done in OCaml. Later also C, Haskell and TypeScript.
- Operates over Integers and Booleans. Later also Tuples and Functions.
- Allows predicates with logical operators, comparisons and addition.

### Example

$$\{(a : \text{Bool}, b : \text{Bool}) \mid (a \vee b) \wedge \neg(a \wedge b)\}$$

$$\{(a : \text{Int}, b : \text{Int}) \mid a \leq b\}$$

# Goals of Thesis

1. Formal language similar to Elm
  - 1.1 A formal syntax
  - 1.2 A formal type system
  - 1.3 A denotational semantic
  - 1.4 A small step semantic (using K Framework) for rapid prototyping the language
  - 1.5 Proof that the type system is valid with respect to the semantics.
2. Extension of the formal language with Liquid Types
  - 2.1 A formal syntax
  - 2.2 A formal type system
  - 2.3 A denotational semantic
  - 2.4 A small step semantic (using K Framework) for rapid prototyping the type checker
  - 2.5 Proof that the extension infers the correct types.
3. A type checker implementation written in Elm for Elm.

# Problems Addressed by the Type System

- Division by zero errors
- Off by one errors
- Proving the correctness of very simple programs
- Clearer interfaces

# Theory: Formalization of the Elm Type System

We will use the Hindley-Milner type system (used in ML, Haskell and Elm)

We say

$T$  is a *mono type*  $\Leftrightarrow T$  is a type variable

$\vee T$  is a type application

$\vee T$  is a algebraic type

$\vee T$  is a product type

$\vee T$  is a function type

$T$  is a *poly type*  $\Leftrightarrow T = \forall a. T'$

where  $T'$  is a mono type or poly type

and  $a$  is a symbol

$T$  is a *type*  $\Leftrightarrow T$  is a mono type  $\vee T$  is a poly type.

## Example

1.  $Nat ::= \mu C. 1 \mid Succ\ C$
2.  $List = \forall a. \mu C. Empty \mid Cons\ a\ C$
3.  $splitAt : \forall a. Nat \rightarrow List\ a \rightarrow (List\ a, List\ a)$

# Theory: Formalization of the Elm Type System

The *values* of a type is the set corresponding to the type:

$$\text{values}(\text{Nat}) = \{1, \text{Succ } 1, \text{Succ Succ } 1, \dots\}$$

$$\text{values}(\text{List Nat}) = \bigcup_{n \in \mathbb{N}} \text{values}_n(\text{List Nat})$$

$$\text{values}_0(\text{List Nat}) = \{[]\}$$

$$\text{values}_n(\text{List Nat}) =$$

$$\{[]\} \cup \{\text{Cons } a \ b \mid a \in \text{values}(\text{Nat}), b \in \text{values}_{n-1}(\text{List Nat})\}$$

# Theory: Definition of Liquid Types

## Definition (Sketch)

Let  $T$  be a Type Application of  $Int$ , tuples and functions. Let  $q$  be a logical formula consisting of

- Logical operations:  $\neg, \wedge, \vee$
- Logical constants:  $True, False$
- Comparisons:  $<, \leq, =, \neq$
- Integer operations:  $+, \cdot c$  where  $c$  is a constant
- Integer constants:  $0, 3, 42, \dots$
- Bound variables:  $a, b, c, \dots$

Then we call the syntactic phrase  $\{a : T | q(a)\}$  a *Liquid Type*.

# Theory: Definition of Liquid Types

## Example

Let  $Nat = \{a : Int \mid a > 0\}$  in

$$\{ \{ (a : Nat, b : Nat) \mid a + b < 42 \} \rightarrow \{ (c : Nat, d : Nat) \mid c \leq d \} \\ \mid (a = c \wedge b = d) \vee (b = c \wedge a = d) \\ \}$$



## Theory: Revisiting the Problems

- Division by zero errors

$$(/) : Int \rightarrow \{a : Int \mid a \neq 0\} \rightarrow Int$$

- Off by one errors

Let  $Pos = \{a : Int \mid 0 \leq a \wedge a < 8\}$  in  
 $get : (Pos, Pos) \rightarrow Chessboard \rightarrow Maybe Figure$

- Proving the correctness of very simple programs

$$swap : \{(a : Int, b : Int) \rightarrow (c : Int, d : Int) \mid b = c \wedge a = d\}$$

- Clearer interfaces

$$length : List\ a \rightarrow \{a : Int \mid a \geq 0\}$$

# Current State

1. Formal language similar to Elm
  - 1.1 A formal syntax **(DONE)**
  - 1.2 A formal type system **(DONE)**
  - 1.3 A denotational semantic **(WORK IN PROGRESS)**
  - 1.4 A small step semantic (using K Framework) for rapid prototyping the language
  - 1.5 Proof that the type system is valid with respect to the semantics.
2. Extension of the formal language with Liquid Types
3. A type checker implementation written in Elm for Elm.

**Started thesis** in July 2019

**Expected finish** at the end of 2020