Refinement Types for Elm

Master Thesis Report

Lucas Payr

30 Oktober 2019

Topics of this Talk

- Introduction To Elm
- Infering the type of an Elm Code
- Introduction to Liquid Types
- Infering Liquid Types

Introduction To Elm

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

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" (Out Of Memory, Stack Overflow, Function Equality)

Theory: Formalization of the Elm Type System

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

We say

$$
\begin{aligned}
T \text{ is a } \textit{mono type} &:\Leftrightarrow T \text{ is a type variable} \\
&\quad \vee\ T \text{ is a type application} \\
&\quad \vee\ T \text{ is a algebraic type} \\
&\quad \vee\ T \text{ is a product type} \\
&\quad \vee\ T \text{ is a function type} \\
T \text{ is a } \textit{poly type} &:\Leftrightarrow T = \forall a.T' \\
&\quad \text{where } T' \text{ is a mono type or poly type} \\
&\quad \text{and } a \text{ is a symbol} \\
T \text{ is a } \textit{type} &:\Leftrightarrow T \text{ is a mono type } \vee \ T \text{ is a poly type.}
\end{aligned}
$$

Theory: Formalization of the Elm Type System

Example

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

Theory: Formalization of the Elm Type System

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

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

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

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

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

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

Order of Types

Let $n, m \in \mathbb{N}$, $T_1, T_2 \in \mathcal{T}$, a_i for all $i \in \mathbb{N}_0^n$ and $b_i \in \mathcal{V}$ for all $i \in \mathbb{N}_0^m$.

We define the partial order $\sqsubseteq$ on poly types as

$$\begin{aligned} \forall a_1 \ldots \forall a_n.T_1 \sqsubseteq \forall b_1 \ldots \forall b_m.T_2 :\Leftrightarrow \\ \exists \Theta = \{(a_i, S_i) \mid i \in \mathbb{N}_1^n \wedge a_i \in \mathcal{V} \wedge S_i \in \mathcal{T}\}. \\ T_2 = [T_1]_\Theta \wedge \forall i \in \mathbb{N}_0^m.b_i \notin \mathrm{free}(\forall a_1 \ldots \forall a_n.T_1) \end{aligned}$$

Given that $T_1 \sqsubseteq T_2$, we say

- T_1 is more general then T_2.
- T_2 is more specific then T_1.

Example: $\forall a.a \sqsubseteq \forall a.List\ a \sqsubseteq List\ Int$

Infering the type of an Elm Code

Infering the Type of the Max Function

```
max : Int -> Int -> Int;
max =
  \a -> \b ->
    if
      (<) a b
    then
      b
    else
      a
```

Infering the Type of the Max Function

$$\frac{a \sqsubseteq_\Delta T}{\Gamma, \Delta \vdash a : T}$$

Instatiation

$$e \sqsubseteq_\Delta T :\Leftrightarrow \exists T_0 \in \mathcal{T}.(e, T_0) \in \Delta \wedge T_0 \sqsubseteq T$$

New rules:

$$\frac{T_0 \sqsubseteq T}{\Gamma, \Delta \cup \{(a, T_0)\} \vdash a : T} \qquad \frac{T_1 \sqsubseteq T}{\Gamma, \Delta \cup \{(b, T_1)\} \vdash b : T}$$

Infering the Type of the Max Function

```
max : Int -> Int -> Int;
max =
  \a -> \b ->
    if
      (<) a b
    then
      b                    --> \forall a.a
    else
      a                    --> \forall a.a
```

Infering the Type of the Max Function

$$\frac{}{\Gamma, \Delta \vdash \text{"}(<)\text{"} : \mathit{Int} \to \mathit{Int} \to \mathit{Bool}}$$

$$\frac{\Gamma, \Delta \vdash e_1 : T_1 \to T_2 \quad \Gamma, \Delta \vdash e_2 : T_1}{\Gamma, \Delta \vdash e_1\ e_2 : T_2}$$

New rule:

$$\frac{\Gamma, \Delta \vdash e_1 : \mathit{Int} \quad \Gamma, \Delta \vdash e_2 : \mathit{Int}}{\Gamma, \Delta \vdash \text{"}(<)\text{"}\ e_1\ e_2 : \mathit{Bool}}$$

Infering the Type of the Max Function

$$\frac{T_0 \sqsubseteq T}{\Gamma, \Delta \cup \{(\mathtt{a}, T_0)\} \vdash \mathtt{a} : T} \qquad \frac{T_1 \sqsubseteq T}{\Gamma, \Delta \cup \{(\mathtt{b}, T_1)\} \vdash \mathtt{b} : T}$$

$$\frac{\Gamma, \Delta \vdash e_1 : \mathit{Int} \quad \Gamma, \Delta \vdash e_2 : \mathit{Int}}{\Gamma, \Delta \vdash \text{"}(<)\text{"}\ e_1\ e_2 : \mathit{Bool}}$$

New rule:

$$\frac{T_0 \sqsubseteq \mathit{Int} \quad T_1 \sqsubseteq \mathit{Int}}{\Gamma, \Delta \cup \{(\mathtt{a}, T_0), (\mathtt{b}, T_1)\} \vdash \text{"}(<)\ \mathtt{a}\ \mathtt{b}\text{"} : \mathit{Bool}}$$

Infering the Type of the Max Function

```
max : Int -> Int -> Int;
max =
  \a -> \b ->
    if
      (<) a b              --> Bool
    then
      b                    --> Int
    else
      a                    --> Int
```

Infering the Type of the Max Function

$$\frac{T_0 \sqsubseteq \mathit{Int} \quad T_1 \sqsubseteq \mathit{Int}}{\Gamma, \Delta \cup \{(a, T_0), (b, T_1)\} \vdash \text{"}(<)\text{"}\ e_1\ e_2 : \mathit{Bool}}$$

$$\frac{\Gamma, \Delta \vdash e_1 : \mathit{Bool} \quad \Gamma, \Delta \vdash e_2 : T \quad \Gamma, \Delta \vdash e_3 : T}{\Gamma, \Delta \vdash \text{"}\texttt{if}\text{"}\ e_1\ \text{"}\texttt{then}\text{"}\ e_2\ \text{"}\texttt{else}\text{"}\ e_3 : T_0}$$

New rule:

$$\frac{T_0 \sqsubseteq \mathit{Int}}{\Gamma, \Delta \cup \{(a, T_0), (b, T_0)\} \vdash \text{"}\texttt{if}(<)\ \texttt{a}\ \texttt{b}\ \texttt{then}\ \texttt{b}\ \texttt{else}\ \texttt{a}\text{"} : T_0}$$

Infering the Type of the Max Function

```
max : Int -> Int -> Int;
max =
  \a -> \b ->
    if                          --> Int
      (<) a b
    then
      b                         --> Int
    else
      a                         --> Int
```

Infering the Type of the Max Function

$$\frac{\Gamma, \Delta \cup \{(a, \overline{\Gamma}(T_1))\} \vdash e : T_2}{\Gamma, \Delta \vdash \text{"}\backslash\text{"}\ a\ \text{"}->\text{"}\ e : T_1 \to T_2}$$

Most General Type

$$\overline{\Gamma} : \Gamma \to \mathcal{T}$$

$$\overline{\Gamma}(T) := \forall a_1 \ldots \forall a_n . T_0$$

such that $\{a_1, \ldots, a_n\} = \text{free}(T') \setminus \{a \mid (a, \_) \in \Gamma\}$

where $a_i \in \mathcal{V}$ for $i \in \mathbb{N}_0^n$ and T_0 is the mono type of T.

We say $\overline{\Gamma}(T)$ is *the most general type* of T.

The most general type of *Int* is *Int*

Infering the Type of the Max Function

Therefore we conclude

$$\frac{}{\Gamma, \Delta \cup \{(a, \mathit{Int})\} \vdash \text{"}\backslash \texttt{b} - > \texttt{if}\ (<)\ \texttt{a}\ \texttt{b}\ \texttt{then}\ \texttt{b}\ \texttt{else}\ \texttt{a}\text{"} : \mathit{Int} \to \mathit{Int}}$$

$$\frac{}{\Gamma, \Delta \vdash \text{"}\backslash \texttt{a} - > \backslash \texttt{b} - > \texttt{if}\ (<)\ \texttt{a}\ \texttt{b}\ \texttt{then}\ \texttt{b}\ \texttt{else}\ \texttt{a}\text{"} : \mathit{Int} \to \mathit{Int} \to \mathit{Int}}$$

Infering the Type of the Max Function

```
max : Int -> Int -> Int;
max =                          --> Int -> Int -> Int
  \a -> \b ->
    if                         --> Int
      (<) a b
    then
      b                        --> Int
    else
      a                        --> Int
```

Introduction to Liquid Types

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

$$\{a : (Bool, Bool) |\ a = (True, False) \vee a = (False, True)\}$$

Background: Liquid Types

Liquid Types (Logically Quantified Data Types) introduced in 2008

- Invented by Patrick Rondan, 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

$$a : Bool \to b : Bool \to \{\nu : Bool | \nu = (a \vee b) \wedge \neg(a \wedge b)\}$$

$$\begin{aligned} a : Int \to b : Int \to \{\nu : Bool \\ | \ (\nu = a \wedge \nu > b) \\ \vee (\nu = b \wedge \nu > a) \\ \vee (\nu = a \wedge \nu = b)\} \end{aligned}$$

$$(/) : Int \to \{\nu : Int | \neg(\nu = 0)\} \to Int$$

Logical Qualifier Expressions

$$
\begin{aligned}
\mathit{IntExp} ::= {} & \mathbb{Z} \\
| {} & \mathit{IntExp} + \mathit{IntExp} \\
| {} & \mathit{IntExp} * \mathbb{Z} \\
| {} & \mathcal{V}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{Q} ::= {} & \mathit{True} \\
| {} & \mathit{False} \\
| {} & \mathit{IntExp} < \mathcal{V} \\
| {} & \mathcal{V} < \mathit{IntExp} \\
| {} & \mathcal{V} = \mathit{IntExp} \\
| {} & \mathcal{Q} \wedge \mathcal{Q} \\
| {} & \mathcal{Q} \vee \mathcal{Q} \\
| {} & \neg \mathcal{Q}
\end{aligned}
$$

Defining Liquid Types

T is a *liquid type* $:\Leftrightarrow$ T is of form $\{a : \mathit{Int} \mid r\}$

where T_0 is a type, a is a symbol, $r \in \mathcal{Q}$,

$\mathit{Nat} := \mu C.1 \mid \mathit{Succ}\ C$

and $\mathit{Int} := \mu\_.0 \mid \mathit{Pos}\ \mathit{Nat} \mid \mathit{Neg}\ \mathit{Nat}$.

$\vee$ T is of form $a : \hat{T}_1 \to \hat{T}_2$

where a is a symbol, $\hat{T}_2$ and $\hat{T}_1$ are liquid types

Infering Liquid Types

Infering the Type of the Max Function

```
max : Int -> Int -> Int;
max =
  \a -> \b ->
    if
      (<) a b
    then
      b
    else
      a
```

Infering the Type of the Max Function

$$\frac{(a, \{\nu : \hat{T} |\ r\}) \in \Delta}{\Gamma, \Delta, \Theta, \Lambda \vdash a : \{\nu : \hat{T} |\ \nu = a\}}$$

New rule:

$$\frac{}{\Gamma, \Delta \cup \{(\mathtt{a}, \{\nu : Int |\ r_0\})\}, \Theta, \Lambda \vdash \mathtt{a} : \{\nu : Int |\ \nu = a\}}$$

$$\frac{}{\Gamma, \Delta \cup \{(\mathtt{b}, \{\nu : Int |\ r_1\})\}, \Theta, \Lambda \vdash \mathtt{b} : \{\nu : Int |\ \nu = b\}}$$

Infering the Type of the Max Function

```
max : Int -> Int -> Int;
max =
  \a -> \b ->
    if
      (<) a b
    then
      b                    --> {v:Int| True }
    else
      a                    --> {v:Int| True }
```

Infering the Type of the Max Function

```
max : Int -> Int -> Int;
max =
  \a -> \b ->
    if
      (<) a b                  --> Bool
    then
      b                        --> {v:Int| True }
    else
      a                        --> {v:Int| True }
```

Infering the Type of the Max Function

$$\frac{}{\Gamma, \Delta \cup \{(a, \{\nu : Int|\ r_0\}), (b, \{\nu : Int|\ r_1\})\}, \Theta, \Lambda \vdash \text{"}(<)\text{"}\ e_1\ e_2 : Bool}$$

$$\frac{\begin{array}{c}\Gamma, \Delta, \Theta, \Lambda \vdash e_1 : Bool \quad e_1 : e_1' \\ \Gamma, \Delta, \Theta, \Lambda \cup \{e_1'\} \vdash e_2 : \hat{T} \quad \Gamma, \Delta, \Theta, \Lambda \cup \{\neg e_1'\} \vdash e_3 : \hat{T}\end{array}}{\Gamma, \Delta, \Theta, \Lambda \vdash \text{"}\texttt{if}\text{"}\ e_1\ \text{"}\texttt{then}\text{"}\ e_2\ \text{"}\texttt{else}\text{"}\ e_3 : \hat{T}}$$

New rule:

$$\frac{\begin{array}{c}\Gamma, \Delta, \Theta, \Lambda \cup \{a < b\} \vdash e_2 : \{\nu : Int|r_0\} \\ \Gamma, \Delta, \Theta, \Lambda \cup \{\neg(a < b)\} \vdash e_3 : \{\nu : Int|r_0\}\end{array}}{\Gamma, \Delta \cup \left\{\begin{array}{l}(a, \{\nu : Int|r_0\}), \\ (b, \{\nu : Int|r_0\})\end{array}\right\}, \Theta, \Lambda \vdash \begin{array}{l}\text{"}\texttt{if}\text{"}\ \texttt{a} < \texttt{b}\ \text{"}\texttt{then}\text{"} \\ \qquad\qquad \texttt{b}\ \text{"}\texttt{else}\text{"}\ \texttt{a}\end{array} : \{\nu : Int|r_0\}}$$

Subtyping

$$\{a_1 : Int | r_1\} <:_{\Theta,\Lambda} \{a_2 : Int | r_2\} \ :\Leftrightarrow$$
$$\text{Let } \{(b_1, T_1), \ldots, (b_n, T_n)\} = \Theta \text{ in}$$
$$\forall k_1 \in \text{value}_\Gamma(T_1). \ldots \forall k_n \in \text{value}_\Gamma(T_n).$$
$$\forall n \in \mathbb{N}. \forall e \in \Lambda.$$
$$[[e]]_{\{(a_1,n),(b_1,k_1),\ldots,(b_n,k_n)\}}$$
$$\wedge [[r_1]]_{\{(a_1,n),(b_1,k_1),\ldots,(b_n,k_n)\}}$$
$$\Rightarrow [[r_2]]_{\{(a_2,n),(b_1,k_1),\ldots,(b_n,k_n)\}}$$

$$a_1 : \hat{T}_1 \to \hat{T}_2 <:_{\Theta,\Lambda} a_2 : \hat{T}_3 \to \hat{T}_4 \ :\Leftrightarrow \ \hat{T}_3 <:_{\Theta,\Lambda} \hat{T}_1 \wedge \hat{T}_2 <:_{\Theta \cup \{(a_1, \hat{T}_3)\},\Lambda} \hat{T}$$

Current State

1. Formal language similar to Elm **(DONE)**
2. Extension of the formal language with Liquid Types
   - 2.1 A formal syntax **(DONE)**
   - 2.2 A formal type system **(WORK IN PROGRESS)**
   - 2.3 Proof that the extension infers the correct types.
3. A type checker implementation written in Elm for Elm.

Started thesis in July 2019

Expected finish in Summer 2021