Refinement Types for Elm

Master Thesis Report

Lucas Payr

10 November 2020

Topics of this Talk

- Introduction To Elm
- Type Inference
- Introduction to Liquid Types
- Liquid Type Inference

Introduction To Elm

Introduction To Elm: 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)
- "No Runtimes errors" (Out Of Memory, Stack Overflow, Function Equality)

Introduction To Elm: Hindley-Milner Type System

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

We say

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

Introduction To Elm: Hindley-Milner Type System

Example

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

Introduction To Elm: Hindley-Milner Type System

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

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

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

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

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

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

Introduction To Elm: 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

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

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

Most General Type

$$
\begin{aligned}
\overline{\Gamma} :& \Gamma \to \mathcal{T} \\
\overline{\Gamma}(T) :=& \forall a_1 \ldots \forall a_n . T_0 \\
& \text{such that } \{a_1, \ldots, a_n\} = \text{free}(T') \setminus \{a \mid (a, \_) \in \Gamma\} \\
& \text{where } a_i \in \mathcal{V} \text{ for } i \in \mathbb{N}_0^n \text{ and } T_0 \text{ is the mono type of } T.
\end{aligned}
$$

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

Type Inference

Type Inference: Infering the Type of the Max Function

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

Type Inference: Infering the Type of the Max Function

$$\frac{(a, \overline{\Gamma}(T)) \in \Delta}{\Gamma, \Delta \vdash a : T}$$

New rules:

$$\frac{}{\Gamma, \Delta \cup \{(a, \overline{\Gamma}(T))\} \vdash a : T} \quad \frac{}{\Gamma, \Delta \cup \{(b, \overline{\Gamma}(T))\} \vdash b : T}$$

Type Inference: Infering the Type of the Max Function

```
max : Int -> Int -> Int;
max =
  \a -> \b ->
    if
      (<) a b
    then
      b                   --> a1
    else
      a                   --> a2
```

Type Inference: Infering the Type of the Max Function

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

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

New rule:

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

Type Inference: Infering the Type of the Max Function

$$\frac{}{\Gamma, \Delta \cup \{(\mathtt{a}, \overline{\Gamma}(T))\} \vdash \mathtt{a} : T} \quad \frac{}{\Gamma, \Delta \cup \{(\mathtt{b}, \overline{\Gamma}(T))\} \vdash \mathtt{b} : T}$$

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

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

New rule:

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

Type Inference: Infering the Type of the Max Function

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

Type Inference: Infering the Type of the Max Function

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

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

New rule:

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

Type Inference: Infering the Type of the Max Function

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

Type Inference: Infering the Type of the Max Function

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

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

Type Inference: Infering the Type of the Max Function

Therefore we conclude

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

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

Type Inference: 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

Introduction to Liquid Types: 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

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

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

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

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

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

Introduction to Liquid Types: Logical Qualifier Expressions

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

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

Introduction to Liquid Types: Defining Liquid Types

$$
\begin{aligned}
T \text{ is a } \textit{liquid type} \;:\Leftrightarrow\;& T \text{ is of form } \{a : \mathit{Int} \mid r\} \\
& \text{where } T_0 \text{ is a type}, a \text{ is a symbol}, r \in \mathcal{Q}, \\
& \mathit{Nat} := \mu C.1 \mid \mathit{Succ}\ C \\
& \text{and } \mathit{Int} := \mu\_.0 \mid \mathit{Pos}\ \mathit{Nat} \mid \mathit{Neg}\ \mathit{Nat}. \\
\vee\; & T \text{ is of form } a : \hat{T}_1 \to \hat{T}_2 \\
& \text{where } a \text{ is a symbol}, \hat{T}_2 \text{ and } \hat{T}_1 \text{ are liquid types}
\end{aligned}
$$

Liquid Type Inference

Liquid Type Inference: Infering the Type of the Max Function

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

Liquid Type Inference: Infering the Type of the Max Function

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

New rule:

$$\frac{\{\nu : \hat{T} \mid \nu = \mathtt{a}\} <:_{\Theta,\Lambda} \{\nu : \hat{T} \mid r\} \quad (\mathtt{a}, \{\nu : \hat{T} \mid r\}) \in \Delta \quad (\mathtt{a}, \{\nu : \hat{T} \mid r\}) \in \Theta}{\Gamma, \Delta, \Theta, \Lambda \vdash \mathtt{a} : \{\nu : \hat{T} \mid \nu = \mathtt{a}\}}$$

$$\frac{\{\nu : \hat{T} \mid \nu = \mathtt{b}\} <:_{\Theta,\Lambda} \{\nu : \hat{T} \mid r\} \quad (\mathtt{b}, \{\nu : \hat{T} \mid r\}) \in \Delta \quad (\mathtt{b}, \{\nu : \hat{T} \mid r\}) \in \Theta}{\Gamma, \Delta, \Theta, \Lambda \vdash \mathtt{b} : \{\nu : \hat{T} \mid \nu = b\}}$$

Liquid Type Inference: Infering the Type of the Max Function

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

Liquid Type Inference: Infering the Type of the Max Function

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

Liquid Type Inference: Infering the Type of the Max Function

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

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

New rule:

$$\frac{\begin{array}{c}\{(a, \{\nu : Int|r_0\}), (b, \{\nu : Int|r_1\})\} \in \Delta \\ \Gamma, \Delta, \Theta, \Lambda \cup \{a < b\} \vdash \texttt{b} : \{\nu : Int|r_2\} \\ \Gamma, \Delta, \Theta, \Lambda \cup \{\neg(a < b)\} \vdash \texttt{a} : \{\nu : Int|r_2\}\end{array}}{\Gamma, \Delta, \Theta, \Lambda \vdash \text{"}\texttt{if}\text{"}\ \texttt{a} < \texttt{b}\ \text{"}\texttt{then}\text{"}\texttt{b}\ \text{"}\texttt{else}\text{"}\ \texttt{a} : \{\nu : Int|r_2\}}$$

Liquid Type Inference: Infering the Type of the Max Function

$$\frac{\{\nu : \hat{T} \mid \nu = \mathtt{a}\} <:_{\Theta,\Lambda} \{\nu : \hat{T} \mid r\} \quad (\mathtt{a}, \{\nu : \hat{T} \mid r\}) \in \Delta \quad (\mathtt{a}, \{\nu : \hat{T} \mid r\}) \in \Theta}{\Gamma, \Delta, \Theta, \Lambda \vdash \mathtt{a} : \{\nu : \hat{T} \mid \nu = \mathtt{a}\}}$$

$$\frac{\{\nu : \hat{T} \mid \nu = \mathtt{b}\} <:_{\Theta,\Lambda} \{\nu : \hat{T} \mid r\} \quad (\mathtt{b}, \{\nu : \hat{T} \mid r\}) \in \Delta \quad (\mathtt{b}, \{\nu : \hat{T} \mid r\}) \in \Theta}{\Gamma, \Delta, \Theta, \Lambda \vdash \mathtt{b} : \{\nu : \hat{T} \mid \nu = b\}}$$

$$\frac{\begin{array}{c}\{(a, \{\nu : Int|r_0\}), (b, \{\nu : Int|r_1\})\} \in \Delta \\ \Gamma, \Delta, \Theta, \Lambda \cup \{a < b\} \vdash \mathtt{b} : \{\nu : Int|r_2\} \\ \Gamma, \Delta, \Theta, \Lambda \cup \{\neg(a < b)\} \vdash \mathtt{a} : \{\nu : Int|r_2\}\end{array}}{\Gamma, \Delta, \Theta, \Lambda \vdash \text{"}\mathtt{if}\text{"}\ \mathtt{a} < \mathtt{b}\ \text{"}\mathtt{then}\text{"}\mathtt{b}\ \text{"}\mathtt{else}\text{"}\ \mathtt{a} : \{\nu : Int|r_2\}}$$

Liquid Type Inference: Infering the Type of the Max Function

Subtyping Rule

$$\frac{\Gamma, \Delta, \Theta, \Lambda \vdash e : \hat{T}_1 \quad \hat{T}_1 <:_{\Theta,\Lambda} \hat{T}_2 \quad \text{wellFormed}(\hat{T}_2, \Theta)}{\Gamma, \Delta, \Theta, \Lambda \vdash e : \hat{T}_2}$$

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

Liquid Type Inference: Infering the Type of the Max Function

Find $r_2 \in \mathcal{Q}$ such that

$$[[((a < b) \land \nu = b) \Rightarrow r_2]]_{\{(a,\{\nu:Int|r_0\}),(b,\{\nu:Int|r_1\})\}}$$

and

$$[[(\neg(a < b) \land \nu = a) \Rightarrow r_2]]_{\{(a,\{\nu:Int|r_0\}),(b,\{\nu:Int|r_1\})\}}$$

are valid.

Use SMT-Solver to find a solution.

Sharpest solution: $r_2 := ((a < \nu \land \nu = b) \lor (\neg(\nu < b) \land \nu = a))$

Liquid Type Inference: Infering the Type of the Max Function

```
max : a:{ v:Int|True } -> b:{ v:Int|True }
  -> { v:Int | (||) ((&&) ((=) v a) ((>) v b))
             ( (||) ((&&) ((=) v b) ((>) v a))
                    ((&&) ((=) v a) ((=) v b))
             ) };

max =
  \a -> \b ->
    if          --> {v:Int
      (<) a b --  | (||) ((&&) ((<) a v) ((=) v b))
    then        --      ((&&) (not ((<) a v)) ((=) v a))
                --  }
      b         --> {v:Int| r_0 }
    else
      a         --> {v:Int| r_1 }
```

Liquid Type Inference: Infering the Type of the Max Function

We infer the type

$$
\begin{aligned}
a : \{\nu : Int | r_0\} \rightarrow & b : \{\nu : Int | r_1\} \\
& \rightarrow \{\nu : Int | (a < \nu \wedge \nu = b) \vee (\neg(\nu < b) \wedge \nu = a)\}
\end{aligned}
$$

The type annotation says the type should be

$$
\begin{aligned}
a : \{\nu : Int | True\} \rightarrow & b : \{\nu : Int | True\} \\
& \rightarrow \{\nu : Int \\
& \mid (a < \nu \wedge \nu = b) \\
& \vee (b < \nu \wedge \nu = a) \\
& \vee (\nu = a \wedge \nu = b)\}
\end{aligned}
$$

Liquid Type Inference: Infering the Type of the Max Function

We set $r_0 = \mathit{True}$, $r_1 = \mathit{True}$ and prove

$$(a < \nu \land \nu = b) \lor (b < \nu \land \nu = a) \lor (\nu = a \land \nu = b)$$

is equivalent to

$$(a < \nu \land \nu = b) \lor (\neg(\nu < b) \land \nu = a)$$

using the Subtype-rule and an SMT-Solver.

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