# Refinement Types for Elm

Master Thesis Report

Lucas Payr

13 April 2021

# Topics of this Talk

- Background
  - Introduction to Elm
  - Introduction to Refinement Types
  - Motivation
  - Goals of the Thesis
- Formulizing the Elm Language
  - Defining the Type System
  - Infering the Type of the Max Function
- Extending the Elm Language
  - Defining Liquid Types
  - Revisiting the Max Function
  - The Inference Algorithm for liquid Types

# Background

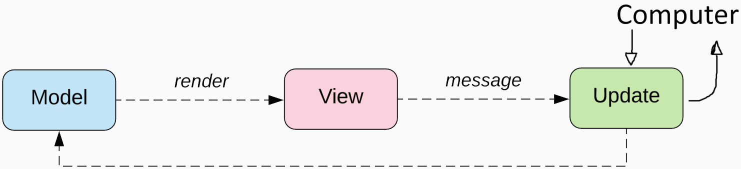
# Background: Introduction to Elm

- Invented by Evan Czaplicki as his master-thesis in 2012.
- 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" (running out of memory, function equality and non-terminating functions still give runtime errors.)

# Background: Introduction to Elm



# Background: Introduction to 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**

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

# Background: Introduction to Refinement 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, Tuples and Functions.
- Allows predicates with logical operators, comparisons and addition.

**Example**

$$a : \{\nu : Int \mid 0 \leq a\} \rightarrow b : \{\nu : Int \mid 0 \leq b\}$$
$$\rightarrow \{\nu : Int \mid 0 \leq \nu \ \wedge \ a - b \leq \nu \ \wedge \ b - a \leq \nu\}$$

## Background: Motivation

- Catching Division by zero in compile time
- Catching index-out-of-bounds errors in compile time
- Having natural numbers as a subtype of ingeters

# Background: Goals of Thesis

1. Formal language similar to Elm
   - A formal syntax
   - A formal type system
   - A denotational semantic
   - A small step semantic (using K Framework) for rapid prototyping the language
   - Proof that the type system is valid with respect to the semantics.
2. Extension of the formal language with Liquid Types
   - Extending the formal syntax, formal type system and denotational semantic
   - Proof that the extension infers the correct types.
   - A Implementation (of the core algorithm) written in Elm for Elm.

# Formulizing the Elm Language

# Formulizing the Elm Language: Defining the Type System

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

We say

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

**Example**

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

# Formulizing the Elm Language: Defining the Type System

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

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

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

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

$$\text{values}_n(\mathit{List}\ \mathit{Nat}) = \{\mathit{Cons}\ a\ b \mid a \in \text{values}(\mathit{Nat}), b \in \text{values}_{n-1}(\mathit{List}\ \mathit{Nat})\}$$

# Formulizing the Elm Language: Defining the Type System

Let $T_1, T_2$ be types.

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

$$T_1 \sqsubseteq T_2 :\Leftrightarrow T_2 \text{ is an instance of } T_1$$

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

# Formulizing the Elm Language: Defining the Type System

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

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

Example: $\forall a . \forall b . \text{List } (a, b)$ is the most general type of $\text{List } (a, b)$.

# Formulizing the Elm Language: The Max Function

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

Starting with leaves of the AST: `a` and `b`.

# Formulizing the Elm Language: The Max Function

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

New rules:

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

# Formulizing the Elm Language: The Max Function

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

Next we derive the type for `(<) a b`.

# Formulizing the Elm Language: 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:

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

# Formulizing the Elm Language: The Max Function

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

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

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

New rule:

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

# Formulizing the Elm Language: The Max Function

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

Next we apply the rule for if-expressions.

# Formulizing the Elm Language: 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:

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

# Formulizing the Elm Language: The Max Function

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

# Formulizing the Elm Language: 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*

# Formulizing the Elm Language: The Max Function

Therefore we conclude

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

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

# Formulizing the Elm Language: The Max Function

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

# Extending the Elm Language

# Extending the Elm Language: Defining Liquid Types

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

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

# Extending the Elm Language: Defining Liquid Types

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

# Extending the Elm Language: Defining Liquid Types

**Subtyping Condition**

We say $c$ is a *Subtyping Condition* $:\Leftrightarrow$ $c$ is of form $T_1 <:_{\Theta,\Lambda} T_2$ where $T_1, T_2$ are a liquid types or templates, $\Theta$ is a type variable context and $\Lambda \subset \mathcal{Q}$.

# Extending the Elm Language: Revisiting the Max Function

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

Again starting at `a` and `b`.

# Extending the Elm Language: Revisiting the Max Function

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

New rule:

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

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

# Extending the Elm Language: Revisiting the Max Function

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

# Extending the Elm Language: Revisiting the Max Function

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

We skip the rule for `(<) a b`: The inferred type is `Bool`

# Liquid Type Inference: Infering the Type of the Max Function

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

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

New rule:

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

# Extending the Elm Language: Revisiting the Max Function

We have yet to provide a judgement for the following rules.

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

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

$$\frac{\{(a, \{\nu : Int \mid r_0\}), (b, \{\nu : Int \mid r_1\})\} \in \Delta \qquad \Gamma, \Delta, \Theta, \Lambda \cup \{a < b\} \vdash \texttt{b} : \{\nu : Int \mid r_2\} \qquad \Gamma, \Delta, \Theta, \Lambda \cup \{\neg(a < b)\} \vdash \texttt{a} : \{\nu : Int \mid r_2\}}{\Gamma, \Delta, \Theta, \Lambda \vdash \texttt{"if" a < b "then"b "else" a} : \{\nu : Int \mid r_2\}}$$

# Extending the Elm Language: Revisiting the Max Function

## Subtyping Rule

$$\frac{\Gamma, \Delta, \Theta, \Lambda \vdash e : T_1 \quad T_1 <:_{\Theta,\Lambda} T_2 \quad \text{wellFormed}(T_2, \Theta)}{\Gamma, \Delta, \Theta, \Lambda \vdash e : T_2}$$

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

# Extending the Elm Language: Revisiting the Max Function

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

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

and

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

are valid.

Use SMT-Solver to find a solution.

Sharpest solution: $r_2 := ((a < \nu \wedge \nu = b) \vee (\neg(\nu < b) \wedge \nu = a))$

# Extending the Elm Language: The Inference Algorithm

We say $T$ is a *template* $:\Leftrightarrow$ $T$ is of form $\{\nu : \mathit{Int} \mid [k]_S\}$

where $k \in \mathcal{K}$ and $S : \mathcal{V} \nrightarrow \mathcal{Q}$

$\vee$ $T$ is of form $a : \{\nu : \mathit{Int} \mid [k]_S\} \to T$

where $k \in \mathcal{K}$, $T$ is a template and $S : \mathcal{V} \nrightarrow \mathit{IntExp}$.

We define $\mathcal{K} := \{\kappa_i \mid i \in \mathbb{N}\}$.

# Extending the Elm Language: The Inference Algorithm

1. (Split) Split the subtyping conditions over dependent function into subtyping conditions over simple liquid types.
2. (Init) Compute $Q = \mathit{Init}(V)$ where $V$ is the set of all occurring variables and initiate the mapping $A$ for very key $\kappa_i$ with the set of resulting predicates with $Q$.
3. (Solve) Check for very subtyping condition if the current mapping $A$ violates the subtyping condition.
4. (Weaken) If so, weaken the mapping by removing any predicate that violates the subtyping condition and repeat
5. One the algorthm terminates we have obtained the strongest refinements that can be build by conjunction over predicates in $Q$.

# Extending the Elm Language: The Inference Algorithm

$$\text{Split} : \mathcal{C} \nrightarrow \mathcal{P}(\mathcal{C}^-)$$

$$\text{Split}(a : \{\nu : \mathit{Int} \mid q_1\} \to T_2 <:_{\Theta,\Lambda} a : \{\nu : \mathit{Int} \mid q_3\} \to T_4) =$$

$$\{\{\nu : \mathit{Int} \mid q_3\} <:_{\Theta,\Lambda} \{\nu : \mathit{Int} \mid q_1\}\} \cup \text{Split}(T_2 <:_{\Theta \cup \{(a,q_3)\},\Lambda} T_4\})$$

$$\text{Split}(\{\nu : \mathit{Int} \mid q_1\} <:_{\Theta,\Lambda} \{\nu : \mathit{Int} \mid q_2\}) =$$

$$\{\{\nu : \mathit{Int} \mid q_1\} <:_{\Theta,\Lambda} \{\nu : \mathit{Int} \mid q_2\}\}$$

$$\mathcal{C} := \{c \mid c \text{ is a subtyping condition}\}$$

$$\mathcal{C}^- := \{ \{\nu : \mathit{Int} \mid q_1\} <:_{\Theta,\Lambda} \{\nu : \mathit{Int} \mid q_2\} \\ \mid (q_1 \in \mathcal{Q} \vee q_1 = [k_1]_{S_1} \text{ for } k_1 \in \mathcal{K}, S_1 \in \mathcal{V} \nrightarrow \mathit{IntExp}) \\ \wedge (q_2 \in \mathcal{Q} \vee q_2 = [k_2]_{S_2} \text{ for } k_2 \in \mathcal{K}, S_2 \in \mathcal{V} \nrightarrow \mathit{IntExp})\}.$$

# Extending the Elm Language: The Inference Algorithm

$$
\begin{aligned}
\mathit{Init} : \mathcal{P}(\mathcal{V}) \to & \mathcal{P}(\mathcal{Q}) \\
\mathit{Init}(V) ::= & \{0 < \nu\} \\
& \cup \{a < \nu \mid a \in V\} \\
& \cup \{\nu < 0\} \\
& \cup \{\nu < a \mid a \in V\} \\
& \cup \{\nu = a \mid a \in V\} \\
& \cup \{\nu = 0\} \\
& \cup \{a < \nu \vee \nu = a \mid a \in V\} \\
& \cup \{\nu < a \vee \nu = a \mid a \in V\} \\
& \cup \{0 < \nu \vee \nu = 0\} \\
& \cup \{\nu < 0 \vee \nu = 0\} \\
& \cup \{\neg(\nu = a) \mid a \in V\} \\
& \cup \{\neg(\nu = 0)\}
\end{aligned}
$$

# Extending the Elm Language: The Inference Algorithm

$\text{Solve} : \mathcal{P}(\mathcal{C}^-) \times (\mathcal{K} \nrightarrow \mathcal{P}(\mathcal{Q})) \rightarrow (\mathcal{K} \nrightarrow \mathcal{P}(\mathcal{Q}))$

$\text{Solve}(C, A) =$

Let $S := \{(k, \bigwedge Q) \mid (k, Q) \in A\}$.

If there exists $(\{\nu : Int \mid q_1\} <:_{\Theta,\Lambda} \{\nu : Int \mid [k_2]_{S_2}\}) \in C$ such that

$\neg(\forall z \in \mathbb{Z}. \forall i_1 \in \text{value}_\Gamma(\{\nu : Int \mid r_1'\}). \ldots \forall i_n \in \text{value}_\Gamma(\{\nu : Int \mid r_n'\}).$

$[[r_1 \wedge p]]_{\{(\nu,z),(b_1,i_1),\ldots,(b_n,i_n)\}} \Rightarrow [[r_2]]_{\{(\nu,z),(b_1,i_1),\ldots,(b_n,i_n)\}})$

then $\text{Solve}(C, \text{Weaken}(c, A))$ else $A$

**SMT statement:**

$$((\bigwedge_{j=0}^{n} [r_j']_{\{(\nu,b_j)\}}) \wedge r_1 \wedge p) \wedge \neg r_2$$

with free variables $\nu \in \mathbb{Z}$ and $b_i \in \mathbb{Z}$ for $i \in \mathbb{N}_1^n$.

# Extending the Elm Language: The Inference Algorithm

$$\text{Weaken} : \mathcal{C}^{-} \times (\mathcal{K} \nrightarrow \mathcal{P}(\mathcal{Q})) \nrightarrow (\mathcal{K} \nrightarrow \mathcal{P}(\mathcal{Q}))$$

$$\text{Weaken}(\{\nu : \mathit{Int} \mid x\} <:_{\Theta,\Lambda} \{\nu : \mathit{Int} \mid [k_2]_{S_2}\}, A) =$$

$$\text{Let } S := \{(k, \bigwedge Q) \mid (k, Q) \in A\},$$

$$Q_2 := \{\ q$$

$$\mid q \in A(k_2)$$

$$\wedge(\forall z \in \mathbb{Z}. \forall i_1 \in \text{value}_\Gamma(\{\nu : \mathit{Int} \mid r_1'\}). \dots . \forall i_n \in \text{value}_\Gamma(\{\nu : \mathit{Int} \mid r_n'\}$$

$$[[r_1 \wedge p]]_{\{(\nu,z),(b_1,i_1),\dots,(b_n,i_n)\}} \Rightarrow [[[q]_{S_2}]]_{\{(\nu,z),(b_1,i_1),\dots,(b_n,i_n)\}})\}$$

$$\text{in } \{(k, Q) \mid (k, Q) \in A \wedge k \neq k_2\} \cup \{(k_2, Q_2)\}$$

**SMT statement:**

$$\neg((\bigwedge_{j=0}^{n} [r_j']_{\{(\nu,b_j)\}}) \wedge r_1 \wedge p) \vee r_2$$

with free variables $\nu \in \mathbb{Z}$ and $b_i \in \mathbb{Z}$ for $i \in \mathbb{N}_1^n$ and $r_2 := [q]_{S_2}$.

# Conclusion

**Positives**

- Can catch index-out-of-bounds errors in compile time
- Can catch (some) division by zero errors in compile time
- Can define the natural numbers as a subtype of the integers.

**Negatives**

- The capabilities of liquid types directly depend on the predicates included in $Init(V)$.
- Increasing the size of $Init(V)$ increases the computation type by a lot. ($\sim$ quadratic time)
- The set of inferrable refinements is always a proper subset of the set of refinements annotatable. Thus, the type system is no longer complete.

# Conclusion

I therefore come to the conclusion, that liquid types are not a proper fit for Elm.

**Started thesis** in July 2019

**Expected finish** in April 2021