Refinement Types for Elm

Master Thesis Report

Lucas Payr

13 April 2021

Topics of this Talk

- Background
  - Introduction to Elm
  - Introduction to Refinement Types
  - Goals of the Thesis
- Formulizing the Elm language
  - Defining the Type system
  - Hindley-Milner Type System
  - Infering the Type of the Max Function
  - Quick introduction to K Framework
- Extending the Elm language
  - Defining Liquid types
  - Liquid Type Inference
  - The Inference Algorithm for Liquid Types
  - Revisiting the Max Function

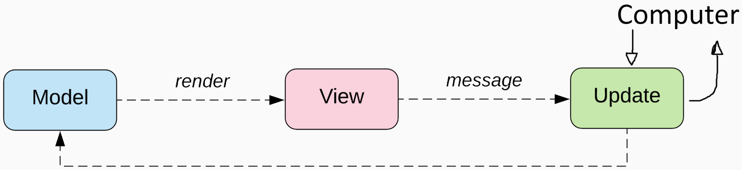
Background: Elm Programming Language

- 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” (running out of memory, function equality and non-terminating functions still give runtime errors.)

Background: The Elm Architecture



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

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

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

Background: Liquid Types

Liquid Types (Logically Quantified Data Types) introduced in 2008

- Invented by Patrick 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

$$a : \{Bool \mid True\} \rightarrow \{\nu : Bool \mid (a \vee \nu) \wedge \neg(a \wedge \nu)\}$$

$$\begin{aligned} a :& \{\nu : Int \mid 0 \leq a\} \rightarrow b : \{\nu : Int \mid 0 \leq b\} \\ & \rightarrow \{\nu : Int \mid 0 \leq \nu \ \wedge \ a - b \leq \nu \ \wedge \ b - a \leq \nu\} \end{aligned}$$

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 Extending the formal syntax, formal type system and denotational semantic
    2.2 Proof that the extension infers the correct types.
    2.3 A Implementation (of the core algorithm) written in Elm for Elm.

Theory: Formalization of the Elm Type System

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

We say

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

Example

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

Theory: Formalization of the Elm Type System

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

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

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

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

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

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

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

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

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

Type Inference: 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

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

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

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

New rule:

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

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

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

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

New rule:

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

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

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

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

Type Inference: Infering the Type of the Max Function

Therefore we conclude

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

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

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
```

K Framework

- 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.

K Framework - K File

```
require "unification.k"
require "elm-syntax.k"

module ELM-TYPESYSTEM
  imports DOMAINS
  imports ELM-SYNTAX

  configuration <k> $PGM:Exp </k>
                <tenv> .Map </tenv>
  //..

  syntax KResult ::= Type
endmodule
```

K Framework - Syntax

`syntax` denotes a syntax

- `strict` - Evaluate the inner expression first
- `right/left` - Evaluate left/right expression first
- `bracket` - Notation for Brackets

```
syntax Type
  ::= "bool"
    | "int"
    | "{}Type"
    | "{" ListTypeFields "}Type" [strict]
    | Type "->" Type        [strict,right]
    | LowerVar
    | "(" Type ")"                [bracket]
    | ..
```

K Framework - Rules

- rules will be executed top to bottom
- `rule . => .` denotes a rewriting rule
- `. ~> .` denotes a concatenation of two processes(KItems)
- `.` denotes the empty process (`rule . ~> A => A`)
- `requires` denotes a precondition to the rule
- `?T` denotes an existentially quantified variable

```
syntax Exp ::= Type
rule E1:Type E2:Type
  => E1 =Type (E2 -> ?T:Type)
     ~> ?T
syntax KResult ::= Type
```

Example for Formally Inferring the Type

```
let
  model = []
in
((::) 1) model
```

0. $\Gamma := \varnothing, \Delta := \varnothing$

[List] 1. $\Gamma, \Delta \vdash [] : \forall a.List\ a$

[LetIn] 2. $\Delta := \Delta \cup (model \mapsto \forall a.List\ a)$

[Int] 3. $\Gamma, \Delta \vdash 1 : Int$

[Call] 4. $\Gamma, \Delta \vdash (::)\ 1 : List\ Int \rightarrow List\ Int$

[Variable] 5. $\Gamma, \Delta \vdash model : \forall a.List\ a$

[Call] 6. $\Gamma, \Delta \vdash ((::)\ 1)\ model : List\ Int$

Formal Inference Rules - List

```
rule []Exp => list ?A:Type

<k>
let
  model = list ?A0:Type
in
((::) (intExp 1)) (variable model)
</k>
<tenv> .Map </tenv>
```

Formal Inference Rules - LetIn

```
rule <k> let X = T:Type in E => E ~> setTenv(TEnv)
  ...</k>
  <tenv> TEnv
    => TEnv[ X
      <- forall
        (#metaKVariables(T)
          -Set #metaKVariables(setTenv(TEnv)))
        .
        ( #freezeKVariables(T, setTenv(TEnv)):>Type)
      ]
  </tenv>
```

```
<k>((::) (intExp 1)) (variable model)</k>
<tenv> [model <- forall A0 . (list (#freeze(A0))]</tenv>
```

Formal Inference Rules - Cons, Int

```
rule (::) => ?A:Type -> ( list ?A ) -> ( list ?A )

rule intExp I:Int => int

<k>
((?A1:Type -> ( list ?A1 ) -> ( list ?A1 )) int)
  (variable model)
</k>
<tenv>
  [model <- forall A0 . (list (#freeze(A0))]
</tenv>
```

Formal Inference Rules - Apply

```
rule E1:Type E2:Type => E1 =Type (E2 -> ?T:Type) ~> ?T

<k>
(( list int ) -> ( list int )) (variable model)
</k>
<tenv>
  [model <- forall A0 . (list (#freeze(A0))]
</tenv>
```

Formal Inference Rules - Variable

```
rule <k> variable X:Id => #renameMetaKVariables(T, Tvs)
    ...</k>
    <tenv>... X |-> forall Tvs . T
    ...</tenv>

<k>
(( list int ) -> ( list int )) (list ?A2)
</k>
<tenv>
  [model <- forall A0 . (list (#freeze(A0))]
</tenv>
```

Formal Inference Rules - Apply

```
<k>
list int
</k>
<tenv>
  [model <- forall A0 . (list (#freeze(A0))]
</tenv>
```

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

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

Introduction to Liquid Types: 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

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

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

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

Introduction to Liquid Types: Logical Qualifier Expressions

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

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

Introduction to Liquid Types: Defining Liquid Types

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

The Inference Algorithm: Definitions

Subtyping Condition

We say c is a *Subtyping Condition* $:\Leftrightarrow$ c is of form $\hat{T}_1 <:_{\Theta,\Lambda} \hat{T}_2$ where $\hat{T}_1, \hat{T}_2$ are a liquid types or templates, Θ is a type variable context and $\Lambda \subset \mathcal{Q}$.

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

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

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

New rule:

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

Liquid Type Inference: Infering the Type of the Max Function

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

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

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

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

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

and

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

are valid.

Use SMT-Solver to find a solution.

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

Template

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

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

The Inference Algorithm

$$
\begin{aligned}
&\mathsf{Infer} : \mathcal{P}(\mathcal{C}) \to \ (\mathcal{K} \nrightarrow \mathcal{Q}) \\
&\mathsf{Infer}(C) = \\
&\quad \text{Let } V := \bigcup_{\hat{T}_1 <:_{\Theta,\Lambda} \hat{T}_2 \in C} \{a \mid (a, \_) \in \Theta\} \\
&\qquad Q_0 := \mathit{Init}(V), \\
&\qquad A_0 := \{(\kappa, Q_0) \mid \kappa \in \bigcup_{c \in C} \mathsf{Var}(c)\}, \\
&\qquad A := \mathsf{Solve}(\bigcup_{c \in C} \mathsf{Split}(c), A_0) \\
&\quad \text{in } \{(\kappa, \bigwedge Q) \mid (\kappa, Q) \in A\}
\end{aligned}
$$

The Inference Algorithm: Step 1 (Split)

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

$$\mathrm{Split}(a : \{\nu : Int | q_1\} \rightarrow \hat{T}_2 <:_{\Theta,\Lambda} a : \{\nu : Int | q_3\} \rightarrow \hat{T}_4) =$$

$$\{\{\nu : Int | q_3\} <:_{\Theta,\Lambda} \{\nu : Int | q_1\}\} \cup \mathrm{Split}(\hat{T}_2 <:_{\Theta \cup \{(a,q_3)\},\Lambda} \hat{T}_4\})$$

$$\mathrm{Split}(\{\nu : Int | q_1\} <:_{\Theta,\Lambda} \{\nu : Int | q_2\}) =$$

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

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

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

The Inference Algorithm: Step 2 (Solve)

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

The Inference Algorithm: Step 2 (Solve)

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

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

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

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

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

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 $i \in \mathbb{N}_1^n$.

The Inference Algorithm: Step 3 (Weaken)

$$
\begin{aligned}
&\text{Weaken} : \mathcal{C}^- \times (\mathcal{K} \nrightarrow \mathcal{P}(\mathcal{Q})) \nrightarrow (\mathcal{K} \nrightarrow \mathcal{P}(\mathcal{Q}))\\
&\text{Weaken}(\{\nu : Int | x\} <:_{\Theta,\Lambda} \{\nu : Int | [k_2]_{S_2}\}, A) =\\
&\quad \text{Let } S := \{(k, \bigwedge Q) \mid (k, Q) \in A\},\\
&\qquad Q_2 := \{\ q\\
&\qquad\qquad \mid q \in A(k_2)\\
&\qquad\qquad \wedge(\forall z \in \mathbb{Z}. \forall i_1 \in \text{value}_\Gamma(\{\nu : Int | r_1'\}). \ldots \forall i_n \in \text{value}_\Gamma(\{\nu : Int | r_n'\}).\\
&\qquad\qquad\quad [[r_1 \wedge p]]_{\{(\nu,z),(b_1,i_1),\ldots,(b_n,i_n)\}} \Rightarrow [[[q]_{S_2}]]_{\{(\nu,z),(b_1,i_1),\ldots,(b_n,i_n)\}})\}\\
&\quad \text{in } \{(k, Q) \mid (k, Q) \in A \wedge k \neq k_2\} \cup \{(k_2, Q_2)\}
\end{aligned}
$$

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 $r_2 := [q]_{S_2}$.

Current State

Started thesis in July 2019

Expected finish in April 2021