4.2 Liquid Types for Elm

We will now extend the type system of Elm with liquid types.

4.2.1 Syntax

We will use the syntax described in the last section.

Definition 4.1: Extended Type Signature Syntax

Given two variable domains <upper-var> and <lower-var>, we define the following syntax:

```
<int-exp-type> ::= Int
                 | <int-exp-type> + <int-exp-type>
                 | <int-exp-type> * Int
                 | 𝒱

<qualifier-type> ::= True
                   | False
                   | (==) <int-exp-type> v
                   | (<) <int-exp-type> v
                   | (<) v <int-exp-type>
                   | (&&) <qualifier-type> <qualifier-type>
                   | (||) <qualifier-type> <qualifier-type>
                   | not <qualifier-type>

<liquid-type> ::= "{v:Int|" <qualifier-type> "}"
                | <lower-var> ":" <liquid-type> "->" <liquid-type>

<type> ::= <liquid-type>
         | "Bool"
         | "List" <type>
         | "(" <type> "," <type> ")"
         | "{" <list-type-fields> "}"
         | <type> "->" <type>
         | <upper-var> <list-type>
         | <lower-var>
```

4.2.2 Type Inference

We will also extend the inference rules. The interesting part is the new judgment for <exp>: We introduce two new set: Θ and Λ. As before, Θ will contain the type of a variable (similarly to the previous section where we Θ stored the value of a variable). Λ contains boolean expressions that get collected while traversing if-then-else branches. We will use these expressions to allow path sensitive subtyping.

TYPE SIGNATURE JUDGMENTS

For type signature judgments, let $exp \in IntExp, q \in \mathcal{Q}$. Let Γ, Δ be type contexts. Let $\Lambda \subset \mathcal{Q}$ and $\Theta : \mathcal{V} \nrightarrow \mathcal{T}$.

For $iet \in$ `"<int-exp-type>"`, the judgment has the form

$$iet : exp$$

which can be read as "*iet* corresponds *exp*".

For $qt \in$ `"<qualifier-type>"`, the judgment has the form

$$qt : q$$

which can be read as "*qt* correspondings to *q*"

For $lt \in$ `"<liquid-type>"`, the judgment has the form

$$lt : \hat{T}$$

which can be read as "*lt* corresponds to the liquid type $\hat{T}$".

As previously already stated, for $t \in$ `<type>` the judgment has the form

$$\Gamma \vdash t : T$$

which can be read as "given Γ, t has the type T".

For $e \in$ `<exp>` the judgment has the form

$$\Gamma, \Delta, \Theta, \Lambda \vdash e : T$$

which can be read as "given Γ, Δ, Θ and Λ, e has the type T".

4.2.3 Auxiliary Definitions

WELL-FORMED LIQUID TYPE

We have already defined well-formed logical qualifiers expressions. We will now extend the notion to well-formed liquid types.

Definition 4.2: Well-formed liquid type

Let $\Theta : \mathcal{V} \nrightarrow \mathcal{T}$.

—

We define following.

$$\text{wellFormed} \subset \{t \in \mathcal{T} | t \text{ is a liquid type}\} \times (\mathcal{V} \nrightarrow \mathbb{N})$$

$$\begin{aligned}
\text{wellFormed}(\{b : \mathit{Int}|r\}, \{(a_1, T_1), \dots, (a_n, T_n)\}) :\Leftrightarrow& \\
\forall k_1 \in \text{value}_\Gamma(T_1). \dots \forall k_n \in \text{value}_\Gamma(T_n).& \\
r \text{ is well defined with respect to } \{(a_1, k_1), \dots, (a_n, k_n)\}&
\end{aligned}$$

$$\text{wellFormed}(a : \hat{T}_1 \to \hat{T}_2, \Theta) :\Leftrightarrow \text{wellFormed}(\hat{T}_1, \Theta) \wedge \text{wellFormed}(\hat{T}_2, \Theta \cup \{(a, \hat{T}_1)\})$$

SUBTYPING

Definition 4.3: Subtyping

Let $\Theta : \mathcal{V} \nrightarrow \mathcal{T}$. Let $\Lambda \subset \mathcal{Q}$, $r_1, r_2 \in \mathcal{Q}$

—

We define the following.

$$\begin{aligned}
\{a_1 : \mathit{Int}|r_1\} <:_{\Theta,\Lambda} \{a_2 : \mathit{Int}|r_2\} \quad :\Leftrightarrow\ & \text{Let } \{(b_1, T_1), \dots, (b_n, T_n)\} = \Theta \text{ in} \\
& \forall k_1 \in \text{value}_\Gamma(T_1). \dots \forall k_n \in \text{value}_\Gamma(T_n). \\
& (\ \forall n \in \mathbb{N}. \forall e \in \Lambda. \\
& \quad [\![e]\!]_{\{(a_1,n),(b_1,k_1),\dots,(b_n,k_n)\}} \\
& \quad \wedge [\![r_1]\!]_{\{(a_1,n),(b_1,k_1),\dots,(b_n,k_n)\}} \wedge [\![r]\!] \\
&) \Rightarrow \forall n \in \mathbb{N}. [\![r_2]\!]_{\Theta \cup \{(a_2,n)\}} \\
a_1 : \hat{T}_1 \to \hat{T}_2 <:_{\Theta,\Lambda} a_2 : \hat{T}_3 \to \hat{T}_4 \quad :\Leftrightarrow\ & \forall n \in \text{value}_{\{\}}(\hat{T}_3). \\
& \hat{T}_1 <:_{\Theta,\Lambda} \hat{T}_3 \wedge \hat{T}_2 <:_{\Theta \cup \{(a_1,n)\},\Lambda} \hat{T}_4
\end{aligned}$$

For two liquid types $\hat{T}_1, \hat{T}_2$, we say $\hat{T}_1$ is a subtype of $\hat{T}_2$ with respect to Θ and Λ if and only if $\hat{T}_1 <:_{\Theta,\Lambda} \hat{T}_2$ is valid.

Subtyping comes with an optional inference rule for `<exp>`. The sharpness of the inferred subtype depends on the capabilities of the SMT-Solver. Using this optional inference rule, the SMT-Solver will need to find the sharpest subtype, or at least sharp enough: In the case of type checking, it might be that the subtype is too sharp and therefore the SMT-Solver can't check the type successfully.

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

Note that we include Λ in our definition. This way we require that the SMT-Solver will allow path sensitive subtyping.

4.2.4 Inference Rules for Type Signatures

INT-EXP-TYPE

Judgment: $iet : exp$

$$\frac{i : Int}{i : i}$$

$$\frac{iet_1 = exp_1 \quad iet_2 = exp_2 \quad exp_1 + exp_2 = exp_3}{iet_1 \texttt{ + } iet_2 = exp_3}$$

$$\frac{i : Int \quad iet = exp_0 \quad exp_0 * i = exp_1}{iet \texttt{ * } i = exp_1}$$

$$\frac{a = exp}{a : exp}$$

QUALIFIER-TYPE

Judgment: $qt : q$

$$\frac{}{\texttt{True} : True}$$

$$\frac{}{\texttt{False} : False}$$

$$\frac{iet : exp_0 \quad exp_0 < \nu = exp_1}{\texttt{(<)}\ iet\ \texttt{v} : exp_1}$$

$$\frac{iet : exp_0 \quad \nu < exp_0 = exp_1}{\texttt{(<)}\ \texttt{v}\ iet : exp_1}$$

$$\frac{iet : exp_0 \quad (\nu = exp_0) = exp_1}{\texttt{(=)}\ \texttt{v}\ iet : exp_1}$$

$$\frac{iet_1 : exp_1 \quad iet_2 : exp_2 \quad exp_1 \land exp_2 = exp_3}{iet_1 \texttt{ \&\& } iet_2 : exp_3}$$

$$\frac{iet_1 : exp_1 \quad iet_2 : exp_2 \quad exp_1 \lor exp_2 = exp_3}{iet_1 \texttt{ || } iet_2 : exp_3}$$

$$\frac{iet : exp_1 \quad \neg exp_1 = exp_2}{\texttt{not}\ iet : exp_2}$$

LIQUID-TYPE

Judgment: $lt : T$

$$\frac{qt : q \quad \{v : Int|\ q\ \} = T}{\texttt{"\{v:Int|"}\ qt\ \texttt{"\}"} : T}$$

$$\frac{lt_1 : T_1 \quad lt_2 : T_2 \quad (a : T_1 \to T_2) = T_3}{a\ \texttt{":"}\ lt_1\ \texttt{"->"}\ lt_2 : T_3}$$

TYPE

Judgment: $\Gamma \vdash t : T$

$$\frac{lt : T}{\Gamma \vdash lt : T}$$

All other inference rules for types have already been described.

4.2.5 Inference Rules for Expressions

EXP

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

Note that we assume that $e_1 \in$ `<qualifier-type>`. If this is not the case, then the judgment can't be derived.

//TODO: Add remaining inference rules.