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
                   | "(<)" v <int-exp-type>
                   | "(==)" v <int-exp-type>
                   | "(&&)" <qualifier-type> <qualifier-type>
                   | "(||)" <qualifier-type> <qualifier-type>
                   | "not" <qualifier-type>

<liquid-type> ::=
       "{v:Int|" <qualifier-type> "}"
       | <lower-var> ":{v:Int|" <qualifier-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 sets: Θ and Λ. As before, Θ will contain the type

of a variable (similarly to the previous section where in Θ we stored the value of a variable). The set 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 $exp \in \mathit{IntExp}$, $q \in \mathcal{Q}$. Let Γ, Δ be type contexts. Let $\Lambda \subset \mathcal{Q}$ and $\Theta : \mathcal{V} \rightharpoonup \mathcal{Q}$.

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

$$iet : exp$$

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

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

$$qt : q$$

which can be read as "qt corresponds to q"

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

$$lt :_\Theta \hat{T}$$

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

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

SUBSTITUTION

Definition 4.2: Refinement Substitution

Let a be a variable, $e \in \mathit{IntExp}$. For $r \in \mathcal{Q}$ let $[r]_{a \leftarrow e}$ denote the substitution on expressions.

—

For a given liquid type we define the *refinement substitution* as follows:

$$[\{b : \mathit{Int} | r\}]_{a \leftarrow e} := \{b : \mathit{Int} | [r]_{a \leftarrow e}\}$$
$$[b : \{c : \mathit{Int} | r\} \to \hat{T}]_{a \leftarrow e} := b : \{c : \mathit{Int} | [r]_{a \leftarrow e}\} \to [\hat{T}]_{a \leftarrow e}$$

Note that we assume for $r \in \mathcal{Q}$ that $[r]_{a \leftarrow e}$ is well formed. In particular, that $[r]_{a \leftarrow e}$ must again life in $\mathcal{Q}$. We can enforce this requirement by the inference rules for `<qualifier-type>`.

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 4.3: Well-formed Liquid Type

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

—

We define following.

$$\begin{aligned}
&\text{wellFormed} \subseteq \{t \in \mathcal{T} | t \text{ is a liquid type}\} \times (\mathcal{V} \rightharpoonup \mathbb{N}) \\
&\text{wellFormed}(\{b : \mathit{Int} | r\}, \{(a_1, r_1), \ldots, (a_n, r_n)\}) :\Leftrightarrow \\
&\qquad \forall k_1 \in \text{value}_\Gamma(\{\nu : \mathit{Int} | r_1\}). \ldots \forall k_n \in \text{value}_\Gamma(\{nu : \mathit{Int} | r_n\}). \\
&\qquad r \text{ is well defined with respect to } \{(a_1, k_1), \ldots, (a_n, k_n), (b, \mathit{Int})\} \\
&\text{wellFormed}(a : \{b : \mathit{Int} | r\} \to \hat{T}, \Theta) :\Leftrightarrow \\
&\qquad (a, _) \notin \Theta \\
&\quad \wedge \text{ wellFormed}(\{b : \mathit{Int} | r\}, \Theta) \\
&\quad \wedge \text{ wellFormed}(\hat{T}, \Theta \cup \{(a, \{b : \mathit{Int} | r\})\})
\end{aligned}$$

SUBTYPING

Definition 4.4: Subtyping

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

—

We define the following.

$$
\begin{aligned}
\{a_1 : Int|q_1\} <:_{\Theta,\Lambda} \{a_2 : Int|q_2\} \ :\Leftrightarrow \\
&\text{Let } \{(b_1, r_1), \ldots, (b_n, r_n)\} = \Theta \text{ in} \\
&\forall k_1 \in \text{value}_\Gamma(\{\nu : Int|r_1\}). \ldots \forall k_n \in \text{value}_\Gamma(\{\nu : Int|r_n\}). \\
&\forall n \in \mathbb{Z}. \forall e \in \Lambda. \\
&\quad [\![e]\!]_{\{(a_1,n),(b_1,k_1),\ldots,(b_n,k_n)\}} \\
&\quad \wedge [\![q_1]\!]_{\{(a_1,n),(b_1,k_1),\ldots,(b_n,k_n)\}} \\
&\Rightarrow [\![q_2]\!]_{\{(a_2,n),(b_1,k_1),\ldots,(b_n,k_n)\}} \\
a_1 : \{b_1 : Int|q_1\} \to \hat{T}_2 <:_{\Theta,\Lambda} a_1 : \{b_2 : Int|q_2\} \to \hat{T}_4 \ :\Leftrightarrow \\
&\{b_2 : Int|q_2\} <:_{\Theta,\Lambda} \{b_1 : Int|q_1\} \\
&\wedge \hat{T}_2 <:_{\Theta \cup \{(a_1, \{b_2 : Int|r_2\})\},\Lambda} \hat{T}_4
\end{aligned}
$$

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 additional 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: $qt : q$

This judgment is used to convert from `qualifier-type` to $\mathcal{Q}$.

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

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

$$\frac{\mathit{iet} : \mathit{exp}_0 \quad \mathit{exp}_0 < \nu = q}{\texttt{(<)}\ \mathit{iet}\ \texttt{v} : q}$$

Note that where we replace the letter `v` with a special character ν.

$$\frac{\mathit{iet} : \mathit{exp}_0 \quad \nu < \mathit{exp}_0 = q}{\texttt{(<)}\ \texttt{v}\ \mathit{iet} : q}$$

$$\frac{\mathit{iet} : \mathit{exp}_0 \quad (\nu = \mathit{exp}_0) = q}{\texttt{(=)}\ \texttt{v}\ \mathit{iet} : q}$$

$$\frac{qt_1 : q_1 \quad qt_2 : q_2 \quad q_1 \wedge q_2 = q_3}{\texttt{(\&\&)}\ qt_1\ qt_2 : q_3}$$

$$\frac{qt_1 : q_1 \quad qt_2 : q_2 \quad q_1 \vee q_2 = q_3}{\texttt{(||)}\ qt_1\ qt_2 : q_3}$$

$$\frac{qt : q_1 \quad \neg q_1 = q_2}{\texttt{not}\ qt : q_2}$$

LIQUID-TYPE

Judgment: $lt :_\Theta \hat{T}$

$$\frac{qt : q \quad \{\nu : \mathit{Int} |\ q\ \} = \hat{T} \quad \text{wellFormed}(\hat{T}_2, \Theta \cup \{(\nu, \mathit{True})\})}{\texttt{"\{v:Int|"}\ qt\ \texttt{"\}"} :_\Theta \hat{T}}$$

$$\frac{\texttt{"\{v:Int|"}\ qt\ \texttt{"\}"} :_\Theta \{\nu : \mathit{Int} |\ q\ \} \quad lt :_{\Theta \cup \{(a,q)\}} \hat{T}_2 \quad (a : \{\nu : \mathit{Int} |\ q\ \} \to \hat{T}_2) = \hat{T}_3}{a\ \texttt{":"}\ \texttt{"\{v:Int|"}\ qt\ \texttt{"\}"}\ \texttt{"->"}\ lt :_\Theta \hat{T}_3}$$

TYPE

Judgment: $\Gamma \vdash t : T$

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

All other inference rules for types have already been described.

4.2.5 Inference Rules for Expressions

EXP

The following are special inference rules for liquid types. For non-liquid types the old rules still apply.

$$\Gamma, \Delta, \Theta, \Lambda \vdash \texttt{"(+)"} : (a : Int \to b : Int \to \{\nu : Int \mid \nu = a + b\})$$

$$\Gamma, \Delta, \Theta, \Lambda \vdash \texttt{"(-)"} : (a : Int \to b : Int \to \{\nu : Int \mid \nu = a + (-b)\})$$

$$\Gamma, \Delta, \Theta, \Lambda \vdash \texttt{"(*)"} : (a : Int \to b : Int \to \{\nu : Int \mid \nu = a * b\})$$

$$\Gamma, \Delta, \Theta, \Lambda \vdash \texttt{"(//)"} : Int \to \{\nu : Int \mid \neg(\nu = 0)\} \to Int$$

By using a liquid type we can avoid dividin by zero.

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

We add the condition e_1 to Λ and ensure that the resulting liquid type is well-formed. Note that we assume that $e_1 \in$ `<qualifier-type>`. If this is not the case, then the inference rule can not be applied and therefore the judgment can not be derived. In some cases we can recover by falling back to the old rule for non-liquid types, but recovery is not guaranteed.

$$\frac{\Gamma, \Delta, \Theta, \Lambda \vdash e_1 : (a : \hat{T}_1 \to \hat{T}_2) \quad \Gamma, \Delta, \Theta, \Lambda \vdash e_2 : \hat{T}_1 \quad e_2 : e_2' \quad [\hat{T}_2]_{a \leftarrow e_2'} = \hat{T}_3}{\Gamma, \Delta, \Theta, \Lambda \vdash e_1\ e_2 : \hat{T}_3}$$

We change the type of e_1 to $a : \hat{T}_1 \to \hat{T}_2$. To ensure that a can't escape the scope, we substitute it with e_2'. Note that we assume that $e_2 \in$ `<qualifier-type>`, else we can try to recover by using the inference rules for non-liquid types.

$$\frac{a : \{\nu : Int | q\} \to \hat{T}_1 = \hat{T}_2 \qquad \Gamma, \Delta \cup \{(a, \{\nu : Int | q\})\}, \Theta \cup \{(a, q)\}, \Lambda \vdash e : \hat{T}_1}{\Gamma, \Delta, \Theta, \Lambda \vdash \texttt{"\textbackslash"}\ a\ \texttt{"->"}\ e : \hat{T}_2}$$

We define the type as $a : \hat{T}_1 \to \hat{T}_2 = \hat{T}_3$. Note that the variable a in the expression realm and the variable a within the context of liquid types are the same. This is because we assume that renaming can be applied at any step of the type inference. To avoid having double bound variables, we require that $a : \hat{T}_1 \to \hat{T}_2$ is well-formed.

$$\frac{(a, \{\nu : Int |\ q\}) \in \Delta \quad (a, q) \in \Theta}{\Gamma, \Delta, \Theta, \Lambda \vdash a : \{\nu : Int |\ \nu = a\}}$$

We can give a variable a sharp liquid type.

—

All other inference rules for expressions have not changed.

4.2.6 Denotational Semantic

For the denotational semantic we only need to extend the semantic for type signatures.

Definition 4.5: Type Signature Semantic

Let Let Γ be a type context. Let $\Theta : \mathcal{V} \nrightarrow \mathcal{Q}$.

—

$$\begin{aligned}
[\![.]\!] :&\texttt{<int-exp-type>} \to \mathit{IntExp} \\
[\![n]\!] =& n \\
[\![iet_1\ \texttt{+}\ iet_2]\!] =& i_1 + i_2 \\
&\text{such that } i_1 = [\![iet_1]\!] \text{ and } i_2 = [\![iet_2]\!] \text{ where } i_1, i_2 \in \mathit{IntExp} \\
[\![iet\ \texttt{*}\ n]\!] =& i \cdot n \\
&\text{such that } i = [\![iet]\!] \text{ where } i \in \mathit{IntExp} \\
[\![a]\!] =& a
\end{aligned}$$

$$[\![.]\!] : \texttt{<qualifier-type>} \to \mathcal{Q}$$

$$[\![\texttt{"True"}]\!] = \mathit{True}$$

$$[\![\texttt{"False"}]\!] = \mathit{False}$$

$$[\![\texttt{"(<)"}\ \mathit{iet}\ v]\!] = i < \nu \quad \text{such that } i = [\![\mathit{iet}]\!] \text{ where } i \in \mathit{IntExp}$$

$$[\![\texttt{"(<)"}\ v\ \mathit{iet}]\!] = \nu < i \quad \text{such that } i = [\![\mathit{iet}]\!] \text{ where } i \in \mathit{IntExp}$$

$$[\![\texttt{"(==)"}\ v\ \mathit{iet}]\!] = \nu = i \quad \text{such that } i = [\![\mathit{iet}]\!] \text{ where } i \in \mathit{IntExp}$$

$$[\![\texttt{"(\&\&)"}\ qt_1\ qt_2]\!] = q_1 \wedge q_2 \quad \text{such that } q_1 = [\![qt_1]\!] \text{ and } q_2 = [\![qt_2]\!] \text{ where } q_1 \in \mathcal{Q} \text{ and } q_2 \in \mathcal{Q}$$

$$[\![\texttt{"(||)"}\ qt_1\ qt_2]\!] = q_1 \vee q_2 \quad \text{such that } q_1 = [\![qt_1]\!] \text{ and } q_2 = [\![qt_2]\!] \text{ where } q_1 \in \mathcal{Q} \text{ and } q_2 \in \mathcal{Q}$$

$$[\![\texttt{"not"}\ qt]\!] = \neg q \quad \text{such that } q = [\![qt]\!] \text{ where } q \in \mathcal{Q}$$

$$[\![.]\!] : \texttt{<liquid-type>} \to \mathcal{T}$$

$$[\![\texttt{"\{v:Int|"}\ qt\ \texttt{"\}"}]\!] = \{\nu : \mathit{Int} |\ r\ \} \quad \text{such that } r = [\![qt]\!] \text{ where } r \in \mathcal{Q}$$

$$[\![a\ \texttt{":"}\ \texttt{"\{v:Int|"}\ qt\ \texttt{"\}"}\ \texttt{"->"}\ lt]\!] = a : \hat{T}_1 \to \hat{T}_2 \quad \text{such that } \hat{T}_1 = [\![\texttt{"\{v:Int|"}\ qt\ \texttt{"\}"}]\!], \text{ and } \hat{T}_2[\![lt]\!] \text{ where } \hat{T}_1, \hat{T}_2 \text{ are liquid types}$$

We extend $[\![.]\!]_\Gamma : \texttt{<type>} \to \mathcal{T}$ by $[\![lt]\!]_\Gamma = [\![lt]\!]$ to now allow liquid types as type signatures.