4 Liquid Types

4.1 Notion of Liquid Types

So-called *refinement types* exclude values from existing types by using a predicate (in this context also called a *refinement*). The definition of such a refinement can be chosen quite freely, but it's important to note that one will also need to provide some inference rules to validate such refinements. This motivates the use of SMT solvers and refinements tailored to the capabilities of specific solvers. Such a set of refinement types are for example *liquid types* (**l**o**g**i**c**ally **qu**al**i**fie**d** data types). We will now specify a version of liquid types. Note that definitions of liquid types vary dependent on the capability of the underlying SMT solver. In our case we will use a very modest definition that should be provable with an arbitrary solver.

We start by defining the syntax and semantic of valid refinements.

Definition 4.1: Logical Qualifier Expressions

We define the set of logical qualifier expressions $\mathcal{Q}$ as follows

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

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

Note that ν is a reserved symbol.

The SMT solver will later solve over ν. The important part to notice is that all logical operators depend on ν whereas the arithmetic operators do not. This is such that the SMT solver can handle these expressions.

Definition 4.2: Well-Formed Logical Qualifier Expressions

Let $e \in \mathcal{Q}$. Let $\Theta : \mathcal{V} \nrightarrow \mathbb{N}$.

—

We say e is *well formed* with respect to Θ iff for all variables v in e,$\Theta(v)$ is well-defined, meaning $\exists n \in \mathbb{N}.(v, n) \in \Theta$.

Variables within the logical qualifier expression are bound by a universal quantifier. Each variable a is bound over a set $\Theta(a)$.

Definition 4.3: Semantics of Logical Qualifier Expressions

We define the semantic of arithmetic expressions *IntExp* as follows.

$$\begin{aligned}
[\![\cdot]\!]_{\cdot} :& \mathit{IntExp} \rightarrow (\mathcal{V} \nrightarrow \mathbb{N}) \rightarrow \mathbb{N} \\
[\![n]\!]_\Theta =& n \\
[\![i+j]\!]_\Theta =& [\![i]\!]_\Theta + [\![j]\!]_\Theta \\
[\![i \cdot n]\!]_\Theta =& [\![i]\!]_\Theta \cdot n \\
[\![a]\!]_\Theta =& \Theta(a)
\end{aligned}$$

Note that we assume that the given expression is well-formed with respect to Θ.

We also define the semantic of logical qualifier expressions $\mathcal{Q}$ as follows:

$$\begin{aligned}
[\![\cdot]\!]_{\cdot} :& \mathcal{Q} \rightarrow (\mathcal{V} \nrightarrow \mathbb{N}) \rightarrow \mathbb{N} \rightarrow \mathit{Bool} \\
[\![\mathit{True}]\!]_\Theta =& \lambda v.\mathit{True} \\
[\![\mathit{False}]\!]_\Theta =& \lambda v.\mathit{False} \\
[\![a \leq \nu]\!]_\Theta =& \lambda v.[\![a]\!]_\Theta \leq v \\
[\![\nu < a]\!]_\Theta =& \lambda v.v < [\![a]\!]_\Theta \\
[\![p \wedge q]\!]_\Theta =& \lambda v.[\![p]\!]_\Theta(v) \wedge [\![q]\!]_\Theta(v) \\
[\![p \vee q]\!]_\Theta =& \lambda v.[\![p]\!]_\Theta(v) \wedge [\![q]\!]_\Theta(v) \\
[\![\neg p]\!]_\Theta =& \lambda v.\neg[\![p]\!]_\Theta(v)
\end{aligned}$$

We will now extend our previous definition of types (see Definition) with the notion of refinement types. This extension is not very interesting, as refinement types don't behave differently from their underlying type.

Definition 4.4: Extended Types

We define the following

T is a *mono type* $:\Leftrightarrow$

$$\begin{aligned}
& T \text{ is a type variable} \\
\vee\ & T \text{ is a type application} \\
\vee\ & T \text{ is a algebraic type} \\
\vee\ & T \text{ is a product type} \\
\vee\ & T \text{ is a function type} \\
\vee\ & T \text{ is a liquid type}
\end{aligned}$$

T is a *poly type* $:\Leftrightarrow$

$$T = \forall a.T'$$
where T' is a mono type
or poly type and a is a symbol.

T is a *type*:$\Leftrightarrow$

$$T \text{ is a mono type}$$
$$\vee\ T \text{ is a poly type.}$$

by using the predicates:

T is a *type variable* $:\Leftrightarrow$ T is a symbol.

T is a *type application* $:\Leftrightarrow$ T is of form $C\ T_1 \dots T_n$
where $n \in \mathbb{N}, C$ is a symbol and the T_i are mono types for all $i \in \mathbb{N}_1^n$.

T is a *algebraic type* $:\Leftrightarrow$ T is of form
$\mu C.C_1\ T_{1,1} \dots T_{1,k(1)} \mid \dots \mid C_n\ T_{n,1} \dots T_{n,k(n)}$
such that $\exists i \in \mathbb{N}.\forall j \in \mathbb{N}_1^{k(i)}.T_{i,j} \neq C$
where $n \in \mathbb{N}, k \in \mathbb{N}_1^n \to \mathbb{N}_0, C$ is a symbol and
$T_{i,k(j)}$ is a mono type
or $T_{i,k(j)} = C$for all $i \in \mathbb{N}_1^n$ and $j \in \mathbb{N}_1^{k(i)}$.

T is a *product type* $:\Leftrightarrow$ T is of form $\{l_1 : T_1, \dots, l_n : T_n\}$
where $n \in \mathbb{N}_0$ and l_i are symbols and T_i are mono types for all $i \in \mathbb{N}_1^n$.

T is a *function type* $:\Leftrightarrow$ T is of form $T_1 \to T_2$
where T_1 and T_2 are mono types.

T is a *liquid type* $:\Leftrightarrow$
T is of form $\{\nu : T_0 \mid r\}$
where T_0 is a type and $r \in \mathcal{Q}$.
$\vee$ T is of form $a : T_1 \to \hat{T}_2$
where $T_1 \in \mathcal{T}$, a is a symbol, $\hat{T}_2$ is a liquid type

We will also need to redefine the definition of free variables and type substitution. The only change is the trival addition of refinement types.

Definition 4.5: Bound, Free, Set of free variables

Let $r \in \mathcal{Q}$, $n \in \mathbb{N}_0$, a be a type variable, T be a type, C be a symbol, $k \in \mathbb{N}_1^n \to \mathbb{N}_0$, T_i be a type, $\hat{T}$ be a liquid type, $T_{i,k(j)}$ be a type or a symbol and C_i be a symbol for all $i \in \mathbb{N}_1^n$ and $j \in \mathbb{N}_1^n$.

—

We say

- a is *free* in $T :\Leftrightarrow a \in \mathit{free}(T)$
- a is *bound* in $T :\Leftrightarrow a \notin \mathit{free}(T)$ and a occurs in T.

where

$$
\begin{aligned}
\text{free}(a) :=& \{a\} \\
\text{free}(C\ T_1 \dots T_n) :=& \bigcup_{i \in \mathbb{N}_1^n} \text{free}(T_i) \\
\text{free}\begin{pmatrix} \mu C. \\ C_1\ T_{1,1} \dots\ T_{1,k(1)} \\ | \dots \\ |\ C_n\ T_{n,1} \dots\ T_{n,k(n)} \end{pmatrix} :=& \bigcup_{i \in \mathbb{N}_0^n} \bigcup_{j \in \mathbb{N}_0^{k_i}} \begin{cases} \varnothing & \text{if } T_{i,j} = C \\ \text{free}(T_{i,j}) & \text{else} \end{cases} \\
\text{free}(\{\_ : T_1, \dots, \_ : T_n\}) :=& \bigcup_{i \in \mathbb{N}_1^n} \text{free}(T_i) \\
\text{free}(T_1 \to T_2) :=& \text{free}(T_1) \cup \text{free}(T_2) \\
\text{free}(\forall a.T) :=& \text{free}(T) \backslash \{a\} \\
\text{free}(\{\nu : T \mid r\}) :=& \text{free}(T) \\
\text{free}(a : T \to \hat{T}) :=& \text{free}(T) \cup \text{free}(\hat{T})
\end{aligned}
$$

Definition 4.6: Type substitution

Let $r \in \mathcal{Q}$, $n \in \mathbb{N}$, $\Theta : \mathcal{V} \nrightarrow \{t \in \mathcal{T} | t \text{ is a mono type}\}$, $a \in \mathcal{V}$. Let $T, T_1, T_2 \in \mathcal{T}$, $k : \mathbb{N}_1^n \to \mathbb{N}_0$ and $T_{i,k(j)} \in \mathcal{T}$ for all $i \in \mathbb{N}_1^n$ and $j \in \mathbb{N}_1^n$. Let $\hat{T}$ be a liquid type.

—

We define the substitute of a type $[.]_\Theta : \mathcal{T} \to \mathcal{T}$ as

$$
\begin{aligned}
[a]_\Theta :=& \begin{cases} S & \text{if } (a, S) \in \Theta \\ a & \text{else} \end{cases} \\
\begin{bmatrix} \mu C. \\ C_1\ T_{1,1} \dots\ T_{1,k(1)} \\ | \dots \\ |\ C_n\ T_{n,1} \dots\ T_{n,k(n)} \end{bmatrix}_\Theta :=& \begin{matrix} \mu C. \\ C_1\ [T_{1,1}]_{\Theta \backslash \{C,\_\}} \dots [T_{1,k_1}]_{\Theta \backslash \{C,\_\}} \\ | \dots \\ |\ C_n\ [T_{n,1}]_{\Theta \backslash \{C,\_\}} \dots [T_{n,k_n}]_{\Theta \backslash \{C,\_\}} \end{matrix} \\
[\{l_1 : T_1, \dots, l_n : T_n\}]_\Theta :=& \{l_1 : [T_1]_\Theta, \dots, l_n : [T_n]_\Theta\} \\
[T_1 \to T_2]_\Theta :=& [T_1]_\Theta \to [T_2]_\Theta \\
[\forall b.T]_\Theta :=& \begin{cases} [T]_\Theta & \text{if } \exists (b, S) \in \Theta \wedge S \notin \mathcal{V} \\ \forall S.[T]_\Theta & \text{if } \exists (b, S) \in \Theta \wedge S \in \mathcal{V} \\ \forall b.[T]_\Theta & \text{else.} \end{cases} \\
[\{\nu : T \mid r\}]_\Theta :=& [T]_\Theta \\
[a : T \to \hat{T}]_\Theta :=& [T]_\Theta \to [\hat{T}]_\Theta
\end{aligned}
$$

Θ is called the set of substitutions.

We will now redefine the notion of values. As mentioned before, liquid types exclude values that do not ensure a specific refinement.

Definition 4.7: Values

Let $r \in \mathcal{Q}$, $\mathcal{S}$ the class of all finite sets, $n \in \mathbb{N}$, $a \in \mathcal{V}$, $T, T_1, T_2, S \in \mathcal{T}$, $k \in \mathbb{N}_1^n \to \mathbb{N}_0$ and $T_{i,k(j)} \in \mathcal{T}$ for all $i \in \mathbb{N}_1^n$ and $j \in \mathbb{N}_1^n$. Let Γ be a type context. Let $\hat{T}$ be a liquid type.

—

We define

$$
\begin{aligned}
\text{values}_\Gamma &: \mathcal{V} \to \mathcal{S} \\
\text{values}_\Gamma(a) &:= \text{values}_\Gamma(\Gamma(a)) \\
\text{values}_\Gamma(C\ T_1\ \ldots\ T_n) &:= \text{values}_\Gamma(\overline{\Gamma(C)}(T_1, \ldots, T_n)) \\
\text{values}_\Gamma\begin{pmatrix} \mu C. \\ |C_1\ T_{1,1} \ldots\ T_{1,k(1)} \\ | \ldots \\ |\ C_n\ T_{n,1} \ldots\ T_{n,k(n)} \end{pmatrix} &:= \bigcup_{i \in \mathbb{N}_0} \text{rvalues}_\Gamma\left(i, \begin{matrix} \mu C. \\ |\ C_1\ T_{1,1} \ldots\ T_{1,k(1)} \\ | \ldots \\ |\ C_n\ T_{n,1} \ldots\ T_{n,k(n)} \end{matrix}\right) \\
\text{values}_\Gamma(\{l_1 : T_1, \ldots, l_n : T_n\}) &:= \Big\{\{l_1 = t_1, \ldots, l_n = t_n\} \\
&\qquad \mid \forall i \in \mathbb{N}_1^n . t_i \in \text{values}_\Gamma(T_i)\Big\} \\
\text{values}_\Gamma(T_1 \to T_2) &:= \{f \mid f \in \text{values}_\Gamma(T_1) \to \text{values}_\Gamma(T_2)\} \\
\text{values}_\Gamma(\forall a.T) &:= \lambda b.\text{values}_{\{(a,b)\} \cup \Gamma}(T) \text{ where the symbol } b \text{ does} \\
&\qquad \text{not occur in } T. \\
\text{values}_\Gamma(\{\nu : T \mid r\}) &:= \text{refinedValues}_{\Gamma,\{\}}(\{\nu : T \mid r\}) \\
\text{values}_\Gamma(a : T \to \hat{T}) &:= \text{refinedValues}_{\Gamma,\{\}}(a : T \to \hat{T})
\end{aligned}
$$

using the following helper functions.

Let $l \in \mathbb{N}, T := \mu C.\ |\ C_1\ T_{1,1} \ldots\ T_{1,k(1)}\ | \ldots\ |\ C_n\ T_{n,1} \ldots\ T_{n,k(n)}$. We define:

$$
\text{rvalues}_\Gamma(0, T) := \left\{ C_i\ v_1 \ldots v_n \;\middle|\; \begin{matrix} i \in \mathbb{N}_1^n \\ \wedge \forall j \in \mathbb{N}_1^{k(i)} . T_{i,j} \neq C \wedge v_j \in \text{values}_\Gamma(T_{i,j}) \end{matrix} \right\}
$$

$$
\text{rvalues}_\Gamma(l+1, T) := \left\{ C_i\ v_1 \ldots v_n \;\middle|\; \begin{matrix} i \in \mathbb{N}_1^n \\ \wedge \forall j \in \mathbb{N}_1^{k(i)} . v_j \in \begin{cases} \text{rvalues}_\Gamma(l, T) & \text{if } T_{i,j} = C \\ \text{values}_\Gamma(T_{i,j}) & \text{else} \end{cases} \end{matrix} \right\}
$$

Let $\Theta : \mathcal{V} \nrightarrow \mathbb{N}$. We define:

$$
\begin{aligned}
&\text{refinedValues}_{\Gamma,\Theta}(\{\nu : T \mid r\}) := \\
&\qquad \{t \in \text{values}_\Gamma(T) \mid r \text{ is well formed with respect to } \Theta \wedge [\![r]\!]_\Theta(t)\} \\
&\text{refinedValues}_{\Gamma,\Theta}(a : T \to \hat{T}) := \\
&\qquad \{b \in \text{values}_\Gamma(T \to \hat{T}) \mid \forall n \in \text{values}_\Gamma(T). b(n) \in \text{refinedValues}_{\Gamma,\Theta \cup \{a,n\}}(\hat{T})\}
\end{aligned}
$$