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**og**i**cally **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.

Example 4.1

$0 \leq \nu \wedge \nu < a + 1$ for $a \in \mathcal{V}$ is a logical qualifier expression. In order to garanty that the SMT solver will find a solution, we require that a is bound with an universal quantifier over a set.

Example 4.2

$0 \leq \nu \wedge b < a + 1$ is not a logical qualifier expression. a and b will be later bound by an universal quantifier, so this expression can be rewritten to either $0 \leq \nu$ or *False*. Both of these rewritten expression are again logical qualifiers.

Similarly, $\nu \leq \nu$ and $\nu < \nu$ are not logical qualifier expression. They can be rewritten to *True* and *False*.

As seen by the examples, we want to ensure that all occurring variables are bound (somewhere outside the expression).

Definition 4.2: Well-Formed Logical Qualifier Expressions

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

—

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

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

Let $\Theta : \mathcal{V} \nrightarrow \mathcal{P}(\mathbb{N})$. Let $a, b \in \mathit{IntExp}$, $n \in \mathbb{N}$, $v \in \mathbb{N}$ and $p, q \in \mathcal{Q}$. Let v be a symbol.

—

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

$$
\begin{aligned}
[\![.]\!] :& \mathit{IntExp} \to \mathbb{N} \to \mathbb{N} \\
[\![n]\!] =& \lambda v.n \\
[\![a + b]\!] =& \lambda v.[\![a]\!](v) + [\![b]\!](v) \\
[\![a \cdot n]\!] =& \lambda v.[\![a]\!](v) \cdot n \\
[\![\nu]\!] =& \lambda v.v
\end{aligned}
$$

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

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

We will now extend the definition of types with the notion of refinement types. This extension is not very interesting, as refinement types don't behave different to their underlying type.

Definition 4.4: Extended Types

We define the following

T is a *mono type* with refinement types :$\Leftrightarrow$

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

T is a *poly type* with refinement types :$\Leftrightarrow$

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

T is a *type* with refinement types:$\Leftrightarrow$

$$\begin{aligned} & T \text{ is a mono type with refinement types} \\ \vee\ & T \text{ is a poly type with refinement types.} \end{aligned}$$

by using the predicates:

$$
\begin{aligned}
T \text{ is a } \textit{type variable} &:\Leftrightarrow T \text{ is a symbol.} \\
T \text{ is a } \textit{type application} &:\Leftrightarrow T \text{ is of form } C\ T_1 \dots T_n \\
&\quad \text{where } n \in \mathbb{N}, C \text{ is a symbol and the } T_i \text{ are mono} \\
&\quad \text{types with refinement types for all } i \in \mathbb{N}_1^n. \\
T \text{ is a } \textit{algebraic type} &:\Leftrightarrow T \text{ is of form} \\
&\quad \mu C.C_1\ T_{1,1} \dots T_{1,k(1)} \mid \dots \mid C_n\ T_{n,1} \dots T_{n,k(n)} \\
&\quad \text{such that } \exists i \in \mathbb{N}. \forall j \in \mathbb{N}_1^{k(i)}. T_{i,j} \neq C \\
&\quad \text{where } n \in \mathbb{N}, k \in \mathbb{N}_1^n \to \mathbb{N}_0, C \text{ is a symbol and} \\
&\quad T_{i,k(j)} \text{ is a mono type with refinement types} \\
&\quad \text{or } T_{i,k(j)} = C \text{for all } i \in \mathbb{N}_1^n \text{ and } j \in \mathbb{N}_1^{k(i)}. \\
T \text{ is a } \textit{product type} &:\Leftrightarrow T \text{ is of form } \{l_1 : T_1, \dots, l_n : T_n\} \\
&\quad \text{where } n \in \mathbb{N}_0 \text{ and } l_i \text{ are symbols and } T_i \text{ are mono} \\
&\quad \text{types with refinement types for all } i \in \mathbb{N}_1^n. \\
T \text{ is a } \textit{function type} &:\Leftrightarrow T \text{ is of form } T_1 \to T_2 \\
&\quad \text{where } T_1 \text{ and } T_2 \text{ are mono types with refinement} \\
&\quad \text{types.} \\
T \text{ is a } \textit{refinement type} &:\Leftrightarrow \\
&\quad T \text{ is of form } \{\nu : T_0 \mid r\} \\
&\quad \text{where } T_0 \text{ is a type with refinement type} \\
&\quad \text{and } r \in \mathcal{Q}. \\
&\vee\ T \text{ is of form } a : T_1 \to T_2 \\
&\quad \text{where } T_1 \in \mathcal{T}, a \text{ is a symbol}, T_2 \text{ is a refinement type}
\end{aligned}
$$

We define $\mathcal{T}_+ := \{T \mid T \text{ is a type with refinement types}\}$ as the set of all types extend with refinement types.

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, $T_{i,k(j)}$ 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

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

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, $k : \mathbb{N}_1^n \to \mathbb{N}_0$ and $T_{i,k(j)} \in \mathcal{T}$ for all $i \in \mathbb{N}_1^n$ and $j \in \mathbb{N}_1^n$.

—

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

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

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, $n \in \mathbb{N}$, $\Theta : \mathcal{V} \nrightarrow \mathcal{T}$, $a \in \mathcal{V}$, $T, T_1, T_2, S \in \mathcal{T}$, $k \in \mathbb{N}_1^n \to \mathbb{N}_0$ 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.

—

We define

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

using the following helper functions.

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

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

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

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

$$\begin{aligned}
\text{refinedValues}_{\Gamma,\Theta}(\{\nu : T \mid r\}) &:= \{b \in \text{values}_\Gamma(T) \\
&\qquad \mid r \text{ is well formed with respect to } \Theta \\
&\qquad \wedge\ \forall (a, X) \in \Theta . \forall a \in X . [\![r]\!](b)\} \\
\text{refinedValues}_{\Gamma,\Theta}(a : T_1 \to T_2) &:= \text{refinedValues}_{\Gamma, \Theta \cup \{a, \text{values}_\Gamma(T_1)\}}(T_2)
\end{aligned}$$