3 Liquid types

First, we define some notations:

- $\mathbb{N}$ are the natural numbers starting from 1.
- $\mathbb{N}_0$ are the natural numbers starting from 0.
- $\mathbb{N}_a^b := \{i \in \mathbb{N}_0 | a \leq i \wedge i \leq b\}$ are the natural numbers between a and b.
- We'll use "." to separate a quantifier from a statement: $\forall a.b$ and $\exists a.b$.
- Function types will be written as $a_1 \to \cdots \to a_n \to b$ instead of $a_1 \times \cdots \times a_n \to b$.
- We allow the use of lambda notation for functions: $\lambda x.x$ instead of $f(x) = x$ for a function f.

3.1 Hindley-Milner type system

For this thesis we will use a Hindley-Milner type system [DM82]. The main idea of the Hindley-Milner type system is to have a type system that implies an order among the types. The ordering will then allow us to infer the type of any expression.

3.1.1 Notion of Types

We will first introduce types, afterwards we will define how types relate to sets by defining the values of types as explicit finite sets. Types are split in *mono types* and *poly types*. Mono types can contain so called *type variables* that can then be bound with a quantifier as a poly type. Note that quantifiers can only occur in the outermost position, thus poly types are more general types than mono types.

Definition 3.1: Mono types, poly types, types

We say

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

by using the following predicates.

$$\begin{aligned}
T \text{ is a } \textit{type variable} &:\Leftrightarrow T \text{ is a symbol.}\\
T \text{ is a } \textit{type application} &:\Leftrightarrow \text{Let } n \in \mathbb{N}, C \text{ be a symbol. Let } T_i \text{ be mono types for all}\\
&\quad\; i \in \mathbb{N}_1^n \text{ in}\\
&\quad\; C\ T_1 \ldots T_n\\
T \text{ is a } \textit{algebraic type} &:\Leftrightarrow \text{Let } n \in \mathbb{N}, k \in \mathbb{N}_1^n \to \mathbb{N}_0, C \text{ be a symbol. Let } T_{i,k(j)} \text{ be a}\\
&\quad\; \text{mono type or } C \text{ for all } i \in \mathbb{N}_1^n \text{ and } j \in \mathbb{N}_1^{k(i)} \text{ in}\\
&\quad\; \mu C.C_1\ T_{1,1} \ldots T_{1,k(1)} \mid \ldots \mid C_n\ T_{n,1} \ldots T_{n,k(n)}\\
&\quad\; \text{such that } \exists i \in \mathbb{N}.\forall j \in \mathbb{N}_1^{k(i)}.T_{i,j} \neq C.\\
T \text{ is a } \textit{product type} &:\Leftrightarrow \text{Let } n \in \mathbb{N}_0. \text{ Let } l_i \text{ symbols and } T_i \text{ be a mono types for}\\
&\quad\; \text{all } i \in \mathbb{N}_1^n \text{ in}\\
&\quad\; T = \{l_1 : T_1, \ldots, l_n : T_n\}\\
T \text{ is a } \textit{function type} &:\Leftrightarrow T_1 \text{ and } T_2 \text{ be mono types in}\\
&\quad\; T = T_1 \to T_2.
\end{aligned}$$

Axiom 3.1

The types T_i for $i \in \mathbb{N}$ in a product type are unordered:

$$\{a : T_1, b : T_2, \ldots\} = \{b : T_2, a : T_1, \ldots\}$$

for any symbols a, b and mono types T_1, T_2.

Example 3.1

The symbol `Char` is a type variable. `Sequence Char` is a type application. They can be thought of as types, whose implementation is unknown. The interpretation of a type variable or a type application depends on its context.

Example 3.2

$\mathit{Bool} = \mu\_.\mathit{True} \mid \mathit{False}$ is an algebraic type.

Example 3.3

$\mathit{List} = \forall a.\mu C.\mathit{Empty} \mid \mathit{Cons}\ a\ C$ is a poly type.

Example 3.4

the empty product type $\{\}$ is a mono type.

Definition 3.2: Sort, Terminal

Let $n \in \mathbb{N}$, $k_j \in \mathbb{N}$, $T_{i,j}$ be a mono type, C, C_i be symbols, $t_j : T_{i,j}$ for all $j \in \mathbb{N}_1^n$, $i \in \mathbb{N}_1^n$. and $T = \mu C.C_1\ T_{1,1} \ldots T_{1,k_1} \mid \ldots \mid C_n\ T_{n,1} \ldots T_{n,k_n}$ be a algebraic type.

—

We call

- $C_i\ T_{i,1} \ldots T_{i,k_1}$ a *sort* of T,
- C_i a *terminal* of T.

Example 3.5

The natural numbers and the integers can be defined as algebraic types using the peano axioms [Pea89]:

- 1 is a natural number.
- Every natural number has a successor.

These axioms can be used for the definition of the type application.

$$Nat ::= \mu C.1 \mid Succ\ C$$

For integers, we can use the property that they contain 0 as well as all positive and negative numbers.

$$Int ::= \mu C.0 \mid Pos\ C \mid Neg\ C$$

In this case numbers like 1, $Succ\ 1$ for Nat or $Neg\ Succ\ 1$ for Int are sorts, where as 1 and $Succ$ for Nat and Neg, Pos and 0 for Int are terminals.

Definition 3.3: Label

Let $n \in \mathbb{N}$. Let T_i be a type, l_i be a unique symbol for all $i \in \mathbb{N}_1^n$.

—

We say l_i are the *labels* of the product type $\{l_1 : T_1, .., l_n : T_n\}$ for all $i \in \mathbb{N}_1^n$.

We define

$$T_1 \times \cdots \times T_n := \{1 : T_1, \ldots, n : T_n\}$$

as the *ordered product type* with n components.

The most general example of a product type is a record. Tuples can be represented as ordered product types.

Definition 3.4: Bound, Free, Set of free variables

Let $n \in \mathbb{N}_0$, 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 $:\Leftrightarrow$ $a \in$ *free*(T)
- a is *bound* in T $:\Leftrightarrow$ $a \notin$ *free* and a occurs in T.

where

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

Definition 3.5: Partial function

Let T_1 and T_2 be types. $f \subseteq T_1 \times T_2$

—

We say f is a *partial function* (Notation: $f : T_1 \nrightarrow T_2$) $:\Leftrightarrow$

$$\forall x \in T_1, y \in T_2.(x, y_1) \in f \wedge (x, y_2) \in f \Rightarrow y_1 = y_2.$$

Definition 3.6: Sets of Types

We define

- $\mathcal{V} := \{a | a \text{ is a symbol}\}$ as the set of all type variables(symbols).
- $\mathcal{T} := \{T | T \text{ is a type}\}$ as the set of all types.

A type can be substituted by replacing a bounded type variable with a mono type:

Definition 3.7: Type substitution

Let $n \in \mathbb{N}$, $\Theta : \mathcal{V} \nrightarrow \mathcal{T}$, $a \in \mathcal{V}$. Let $T, T_1, T_2, S \in \mathcal{T}$, $k \in \mathbb{N}_1^n \to \mathbb{N}_0$ and $T_{i,k(j)} \in \mathcal{T}$ for all $i \in \mathbb{N}_1^n$ and $j \in \mathbb{N}_1^n$.

—

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

$$[a]_\Theta := \begin{cases} S & \text{if } (a, S) \in \Theta \\ a & \text{else} \end{cases}$$

$$\begin{bmatrix} \mu C. \\ C_1\ T_{1,1} \dots\ T_{1,k(1)} \\ | \dots \\ |\ C_n\ T_{n,1} \dots\ T_{n,k(n)} \end{bmatrix}_\Theta := \begin{matrix} \mu C. \\ C_1\ [T_{1,1}]_\Theta \dots [T_{1,k_1}]_\Theta \\ | \dots \\ |\ C_n\ [T_{n,1}]_\Theta \dots [T_{n,k_n}]_\Theta \end{matrix}$$

$$[\{l_1 : T_1, \dots, l_n : T_n\}]_\Theta := \{l_1 : [T_1]_\Theta, \dots, l_n : [T_n]_\Theta\}$$

$$[T_1 \to T_2]_\Theta := [T_1]_\Theta \to [T_2]_\Theta$$

$$[\forall b.T]_\Theta := \begin{cases} [T]_\Theta & \text{if } \exists (b, \_) \in \Theta \\ \forall b.[T]_\Theta & \text{else.} \end{cases}$$

Θ is called the set of substitutions.

The type substitution gives raise to a partial order $\sqsubseteq$:

Definition 3.8: Type Order

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 $\sqsubseteq$ as

$$\forall a_1 \dots \forall a_n.T_1 \sqsubseteq \forall b_1 \dots \forall b_m.T_2 :\Leftrightarrow \exists \Theta = \{(a_i, S_i) | i \in \mathbb{N}_1^n \wedge a_i \in \mathcal{V} \wedge S_i \in \mathcal{T}\}.$$
$$T_2 = [T_1]_\Theta \wedge \forall i \in \mathbb{N}_0^m.b_i \notin \text{free}(\forall a_1 \dots \forall a_n.T)$$

The rule can be read as follows:

- First replace all bounded variables with types.
- Next rebound any new variables (variables that were previously not free).

Example 3.6

$\forall a.a$ is the smallest type in the type system. The partial order forms a tree structure with $\forall a.a$ at the root and different branches for $\forall a.\forall b.(a, b)$, $\forall a.C\ a$, $\forall a.\forall b.a \to b$ and so on. The leaves of the tree are all possible mono types.

3.1.2 Interpretation of types

Before we interprete a type, we will first introduce a set of labeled elements as a record.

Definition 3.9: Record

Let n in $\mathbb{N}$, l_i be a symbol, t_i arbitary for all i in $\mathbb{N}_1^n$.

—

We define

$$\{l_1 = t_1, \dots, l_n = t_n\} : \{l_1, \dots, l_n\} \to \{t_1, \dots, t_n\}$$
$$\{l_1 = t_1, \dots, l_n = t_n\}(l) := t \text{ such that } \exists i \in \mathbb{N}_1^n . l = l_i \wedge t = t_i$$

Note that values of a ordered product type are equivant to tuples:

$$\forall i \in \mathbb{N}_1^n . \{1 = t_1, \dots, n = t_n\}(i) = (t_1, \dots, t_n).i$$

Thus we will use the notation of tuples for values of a ordered product type.

Definition 3.10: Application Constructor

Let $n \in \mathbb{N}_0$. Let a_i be a symbol for all $i \in \mathbb{N}_1^n$.

—

We call

$$f : \underbrace{\mathcal{T} \to \cdots \to \mathcal{T}}_{n \text{ times}} \to \mathcal{T}$$
$$f(T_1, \dots, T_n) := [\forall a_1 \dots a_n . T]_{\{(a_1, T_1), \dots, (a_n, T_n)\}} \text{ for a mono type } T.$$

an *application constructor*.

We define $\mathcal{C} = \{f | f \text{ is a application constructor}\}$ as the set of all application constructors.

Definition 3.11: Context

$\Gamma : \mathcal{V} \nrightarrow \mathcal{C}$ is a *context* $:\Leftrightarrow$

$\Gamma = \{\}$
$\vee\ \Gamma = \Delta \cup \{(a, T)\}$ where T is a mono type, Δ is a context and a is a type variable
$\vee\ \Gamma = \Delta \cup \{(C, f)\}$ where $T = C\ T_1 \dots T_n$ is a type application and T_i is a mono type for $i \in \mathbb{N}_1^n$ and f is a application constructor such that $f(T_1, \dots, T_n)$ is a mono type.

Definition 3.12: Values

Let $\mathcal{S}$ 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 context.

—

We define

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

using the following helper function.

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

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

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

The base case of this recursive function is in rvalue$(0, T)$ for a given T.

Theorem 3.1: rvalue is nested

Let $n \in \mathbb{N}_0$. Let $T := \mu C. \mid C_1\ T_{1,1} \dots\ T_{1,k(1)} \mid \dots \mid C_n\ T_{n,1} \dots\ T_{n,k(n)}$.

—

$$
\bigcup_{i \in \mathbb{N}_0^n} \text{rvalues}_\Gamma(i, T) = \text{rvalues}_\Gamma(n, T)
$$

Proof. Its sufficient to prove by induction over n that

$$
\forall n \in \mathbb{N}_0 . \text{rvalues}_\Gamma(n, T) \subseteq \text{rvalues}_\Gamma(n+1, T).
$$

Base case: We'll show $\text{rvalues}_\Gamma(0,T) \subseteq \text{rvalues}_\Gamma(1,T)$.

$$\begin{aligned}
&\text{rvalues}_\Gamma(1,T)\\
&= \left\{ C_i\ v_1 \ldots v_n \;\middle|\; \begin{array}{l} i \in \mathbb{N}_1^n \\ \wedge \forall j \in \mathbb{N}_1^{k(i)}.v_j \in \begin{cases} \text{rvalues}_\Gamma(0,T) & \text{if } T_{i,j} = C \\ \text{values}_\Gamma(T_{i,j}) & \text{else} \end{cases} \end{array} \right\}\\
&\supseteq \left\{ C_i\ v_1 \ldots v_n \;\middle|\; \begin{array}{l} i \in \mathbb{N}_1^n \\ \wedge \forall j \in \mathbb{N}_1^{k(i)}.T_{i,j} \neq C \wedge v_j \in \text{values}_\Gamma(T_{i,j}) \end{array} \right\}\\
&= \text{rvalues}_\Gamma(0,T).
\end{aligned}$$

Inductive step: Assuming $\text{rvalues}_\Gamma(n,T) \subseteq \text{rvalues}_\Gamma(n+1,T)$, we'll prove

$$\text{rvalues}_\Gamma(n+1,T) \subseteq \text{rvalues}_\Gamma(n+2,T).$$

$$\begin{aligned}
&\text{rvalues}_\Gamma(n+2,T)\\
&= \left\{ C_i\ v_1 \ldots v_n \;\middle|\; \begin{array}{l} i \in \mathbb{N}_1^n \\ \wedge \forall j \in \mathbb{N}_1^{k(i)}.v_j \in \begin{cases} \text{rvalues}_\Gamma(n+1,T) & \text{if } T_{i,j} = C \\ \text{values}_\Gamma(T_{i,j}) & \text{else} \end{cases} \end{array} \right\}\\
&\supseteq \left\{ C_i\ v_1 \ldots v_n \;\middle|\; \begin{array}{l} i \in \mathbb{N}_1^n \\ \wedge \forall j \in \mathbb{N}_1^{k(i)}.v_j \in \begin{cases} \text{rvalues}_\Gamma(n,T) & \text{if } T_{i,j} = C \\ \text{values}_\Gamma(T_{i,j}) & \text{else} \end{cases} \end{array} \right\}\\
&= \text{rvalues}_\Gamma(n+1,T)
\end{aligned}$$

□

As an example we can now prove that the values of Nat from example are isomorphic to the natural numbers.

Theorem 3.2

Let $Nat := \mu C.1|Succ\ C$

—

$$\text{values}(Nat) \cong \mathbb{N}$$

Proof.

$$\begin{aligned}
\text{value}(\mu C.1|Succ\ C) &= \bigcup_{i \in \mathbb{N}_0} \text{rvalue}(i, \mu C.1|Succ\ C)\\
&= \lim_{n\to\infty} \bigcup_{i \in \mathbb{N}_0^n} \text{rvalue}(i, \mu C.1|Succ\ C)\\
&= \lim_{n\to\infty} \text{rvalue}(n, \mu C.1|Succ\ C).
\end{aligned}$$

We'll now show by induction over $n \in \mathbb{N}_0$ that

$$\text{rvalue}(n, \mu C.1|Succ\ C) = \{1, \underbrace{Succ\ 1, \ldots, Succ\ \ldots\ Succ\ 1}_{n \text{ times}}\}.$$

Base case: $rvalue(0, \mu C.1|Succ\ C) = \{1\}$
Inductive step:
Assuming rvalue$(n, \mu C.1|Succ\ \ C) = \{1, \underbrace{Succ\ 1, \ldots, Succ\ \ldots\ Succ\ 1}_{n \text{ times}}\}$, we'll prove rvalue$(n+1, \mu C.1|Succ\ C) = \{1, \underbrace{Succ\ 1, \ldots, Succ\ \ldots\ Succ\ 1}_{n+1 \text{ times}}\}$.

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

We define a order on

$$\text{value}(Nat) = \lim_{n \to \infty} \{1, \underbrace{Succ\ 1, \ldots, Succ\ \ldots\ Succ\ 1}_{n \text{ times}}\} = \{1, Succ\ 1, \ldots, Succ\ Succ\ 1, \ldots\}$$

by $v_1 < v_2 :\Leftrightarrow v_2 = Succ \ldots Succ\ v_1$ for $v_1, v_2 \in$ value(Nat). This is a well-ordering, thus the set is isomorphic to $\mathbb{N}$. $\square$

References

[DM82] Luís Damas and Robin Milner. "Principal Type-Schemes for Functional Programs". In: *Conference Record of the Ninth Annual ACM Symposium on Principles of Programming Languages, Albuquerque, New Mexico, USA, January 1982*. 1982, pp. 207–212. DOI: 10.1145/582153.582176. URL: https://doi.org/10.1145/582153.582176.

[Pea89] G. Peano. *Arithmetices principia: nova methodo*. Trans. by Vincent Verheyen. Fratres Bocca, 1889. URL: https://github.com/mdnahas/Peano_Book/blob/master/Peano.pdf.