# 3 Liquid Types

## 3.1 Defining the Type System

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 $\exists a.b$.
- Functions will be written as $a_1 \to \cdots \to a_n \to b$ instead of $a_1 \times \cdots \times a_n \to b$.

For this thesis we will use a Hindley-Milner type system [DM82].

**Definition 3.1: Types**

We say

$$\begin{aligned} T \text{ is a } \textit{type} :\Leftrightarrow & T \text{ is a mono type} \\ & \vee T \text{ is a poly type} \end{aligned}$$

We write $v : T$ to declare that $v$ has the type $T$.

**Definition 3.2: Mono types, Poly types**

Let $n \in \mathbb{N}$.

—

We say

$$\begin{aligned} T \text{ is a } \textit{mono type} :\Leftrightarrow & T \text{ is a symbol} \\ & \vee T ::= C_1\ T_{1,1} \ldots T_{1,k_1} \mid \ldots \mid C_n\ T_{n,1} \ldots T_{n,k_n} \\ & \vee T ::= \{l_1 : T_1, .., l_n : T_n\} \\ & \vee T ::= T_1 \to T_2 \\ T \text{is a } \textit{poly type} :\Leftrightarrow & T ::= \forall a.T' \end{aligned}$$

where $T_1, T_2, T_i, T_{i,j}$ are a mono types and $l_i, C_i$ are symbols for all $i \in \mathbb{N}_1^n$ and $j \in \mathbb{N}_0^{k_i}$; $T'$ is a type and $a$ is a symbol in $T'$.

**Definition 3.3: Type variable**

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

**Example 3.1**

The symbol `string` is a type variable. It can be though of as a type, whose implementation is unknown. For real programming languages this is not allowed.

**Definition 3.4: Type application, Sort, Value, Constructor**

Let $n \in \mathbb{N}$, $k_j \in \mathbb{N}$ $T_{i,j}$ be a mono type $C_i$ be a symbol and $t_j : T_{i,j}$ for all $j \in \mathbb{N}_1^n$ and $i \in \mathbb{N}_1^n$.

—

We call $T ::= C_1\ T_{1,1} \ldots T_{1,k_1} \mid \ldots \mid C_n\ T_{n,1} \ldots T_{n,k_n}$ a *type application*.

For a give type application we call

- $C_i\ T_{i,1} \ldots T_{i,k_1}$ a *sort* of $T$,
- $C_i\ t_0 \ldots t_{k_i}$ a *value* of $T$,
- $C_i$ a *terminal* of $T$,
- The function
$$C_i : T_{i,1} \to \cdots \to T_{i,k_1} \to T$$
$$C_i(t_1, \ldots, t_n) := C_i\ t_1 \ldots\ t_n$$
  a *constructor* of $T$.

Note: The vertical line "|" can be read as "or". It's not a predicate but rather separates the sorts from each other.

**Example 3.2**

*Bool* ::= *True* | *False* is a type application.

**Example 3.3**

The natural numbers and the integers can be defined as type applications 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.

$$Nat ::= 1 \mid Succ\ Nat$$

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

$$Int ::= 0 \mid Pos\ Nat \mid Neg\ Nat$$

**Definition 3.5: Product type**

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

—

We call $T = \{l_1 : T_1, .., l_n : T_n\}$ a *product type*.

- We call $l_i$ the *labels* of the product type for all $i \in \mathbb{N}_1^n$.

- The values of a product type have the form $\{l_1 = t_1, \dots, l_n = t_n\}$ where $t_i : T_i$ for all $i \in \mathbb{N}_1^n$.
- The types $T_i$ are unordered: $\{a : T_1, b : T_2\} = \{b : T_2, a : T_1\}$.

For ordered product types we write

$T_1 \times \dots \times T_n := \{1 : T_1, \dots, n : T_n\}$.

Values of a ordered product type have the form $(t_1, \dots, t_n)$, where $t_i : T_i$ for all $i \in \mathbb{N}_1^n$.

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

**Definition 3.6: Functions**

Let $T_1$ and $T_2$ be types.

—

We call $T_1 \to T_2$ a *function*. A function type acts exactly the same as a function over sets.

**Example 3.4**

Let $T_1, T_2, T_3$ be types.

—

Then $(T_1 \times T_2 \to T_3)$ is isomorphic to $T_1 \to (T_2 \to T_3)$. This was originally proven by Gottlob Frege [Sch24]. This method for translating multivariable functions into single variable functions is nowadays called *currying* and named after Haskell Curry who further developed the theory [CF58].

**Definition 3.7: Bounded, Free, Set of free variables**

Let $n \in \mathbb{N}$, $a$ be a type variable, $T$ be a type, $k \in \mathbb{N}_1^n \to \mathbb{N}_0$, and $T_{i,k(j)}$ be a type for all $i \in \mathbb{N}_1^n$ and $j \in \mathbb{N}_1^n$.

—

$a \in \mathcal{V}$ is called *bounded*. Unbounded type variables are called *free*.

The set of all free type variables of a type is defined as follows:

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

**Example 3.5**

Let $T ::= C\ a$ be a type application.

—

We will later see that $a$ may be substituted by the quantified type $\forall a.a$. This would lead to $T ::= C\ (\forall a.a)$, but as quantifiers always move to the upper most level, it results in $\forall a.T ::= C\ a$ instead.

**Example 3.6**

$\forall a.List\ a ::= Empty \mid Cons\ a\ (List\ a)$ is a poly type.

**Definition 3.8: partial function**

Let $T_1, T_2$ be types.

—

We define $f \subseteq T_1 \times T_2$ as a partial function from $T_1$ to $T_2$.

When ever we write $f \subseteq T_1 \times T_2$, we assume that f is univariant:

$$(x, y_1) \in f \wedge (x, y_2) \in f \Rightarrow y_1 = y_2$$

**Definition 3.9: Sets of Types**

We define

- $\mathcal{V} := \{a | a \text{ is a type variable}\}$ as the set of all type variables, evidently the set of all 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 3.10: Type substitution**

Let $n \in \mathbb{N}$, $\quad \Theta \subseteq \mathcal{V} \times \mathcal{T}$, $\quad a \in \mathcal{V}$, $\quad T, T_1, T_2, S \in \mathcal{T}$, $\quad k \in \mathbb{N}_1^n \to \mathbb{N}_0$, 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

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

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

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

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

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

$\Theta$ is called the set of substitutions.

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

**Axiom 3.1: Type Order**

Let $n, m \in \mathbb{N}$ with $m \leq n$, $T_1, T_2 \in \mathcal{T}$, $a_i, b_i \in \mathcal{V}$, $S_i \in \mathcal{T}$, for all $i \in \mathbb{N}_0^n$ and $\Theta = \bigcup\{(a_i, S_i)\}$

We define the partial order $\sqsubseteq$ such that

$$\frac{T_2 = [T_1]_\Theta \quad \forall i \in \mathbb{N}_0^m . b_i \notin \text{free}(\forall a_1 \dots \forall a_n . T) \quad m \leq n}{\forall a_1 \dots \forall a_n . T_1 \sqsubseteq \forall b_1 \dots \forall b_m . T_2}$$

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).

## References


[CF58] H.B. Curry and R. Feys. *Combinatory Logic*. Combinatory Logic v. 1. North-Holland Publishing Company, 1958. URL: https://books.google.at/books?id=fEnuAAAAMAAJ.

[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.

[Sch24] M. Schönfinkel. "Über die Bausteine der mathematischen Logik". In: *Mathematische Annalen* 92.3 (Sept. 1924), pp. 305–316. ISSN: 1432-1807. DOI: 10.1007/BF01448013. URL: https://doi.org/10.1007/BF01448013.