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

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 outer most position thus poly types are more general types than mono types.

Definition 3.1: Mono types, Poly types, types

We say

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

by using the following predicates.

Let $n \in \mathbb{N}$, $k \in \mathbb{N}_1^n \to \mathbb{N}_0$, $T_1, T_2, T_i, T_{i,j}$ are a mono types and C_i, l_i are symbols for all $i \in \mathbb{N}_1^n$ and $j \in \mathbb{N}_0^{k_i}$.

$$\begin{aligned} T \text{ is a } \textit{type variable} :\Leftrightarrow & T \text{ is a symbol.} \\ T \text{ is a } \textit{algebraic type} :\Leftrightarrow & T ::= C_1\ T_{1,1} \ldots T_{1,k_1} \mid \ldots \mid C_n\ T_{n,1} \ldots T_{n,k_n} \\ T \text{ is a } \textit{product type} :\Leftrightarrow & T ::= \{l_1 : T_1, \ldots, l_n : T_n\} \\ T \text{ is a } \textit{function type} :\Leftrightarrow & T ::= T_1 \to T_2. \end{aligned}$$

Example 3.1

The symbol `string` is a type variable. It can be thought of as a type, whose implementation is unknown. For real programming languages, a set of type-variables with fixed interpretations is assumed.

Example 3.2

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

Definition 3.2: 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, $t_j : T_{i,j}$ for all $j \in \mathbb{N}_1^n$, $i \in \mathbb{N}_1^n$. and $T ::= C_1\ T_{1,1} \dots T_{1,k_1} \mid \dots \mid C_n\ T_{n,1} \dots T_{n,k_n}$ be a algebraic type.
—

We call

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

Example 3.3

$Bool ::= True \mid False$ is an algebraic type, $True$ and $False$ are terminals. In this case the sorts of the type correspond with the terminals.

Example 3.4

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.

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

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

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

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

Definition 3.3: Product type

Let $n \in \mathbb{N}$, T_i be a type, l_i be a unique symbol for all $i \in \mathbb{N}_1^n$, $T = \{l_1 : T_1, .., l_n : T_n\}$ be a product type.
—

We say

- l_i are the *labels* of the product type 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

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

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

Definition 3.4: 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 $j \in \mathbb{N}_1^n$.
—

We say

- a is *bound* in $\forall b.T :\Leftrightarrow \begin{cases} a = b & \text{if } T \text{ is a mono type} \\ a = b \vee a \text{ is bound in } T & \text{if } T \text{ is a poly type} \end{cases}$
- a is *free* in $\forall b.T :\Leftrightarrow a \neq b \wedge a \in \mathit{free}(T)$

where

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

Definition 3.5: Partial function

Let T_1, T_2 be types.
—

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

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

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

Definition 3.6: Sets of Types

We define

- $\mathcal{V} := \{a | a \text{ is a type variable}\}$ 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 3.7: Type substitution

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

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

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

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

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

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

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

Θ is called the set of substitutions.

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

Definition 3.8: Type Order

Let $n, m \in \mathbb{N}$ 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

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

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).

3.1.2 Interpretation of types

First we define the notion of 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 $\{l_1 = t_1, \ldots, l_n = t_n\} := A$ such that $A = \{t_1, ..., t_n\}$ and $A.l_i := t_i$ for all $i \in \mathbb{N}_1^n$. Note that values of ordered product types are tuples:

$$1 = t_1, \ldots, n = t_n = (t_1, \ldots, t_n)$$

Next we define the values of a type.

Definition 3.10: Value

Let Ω the class of all finite sets, $n \in \mathbb{N}$, $\Theta \subseteq \mathcal{V} \times \mathcal{T}$, $a \in \mathcal{V}$, $T, T_1, T_2, S \in \mathcal{T}$, $k \in \mathbb{N}_1^n \to \mathbb{N}_0$ and $T_{i,k(j)} \in \mathcal{T}$ for all $i \in \mathbb{N}_1^n$ and $j \in \mathbb{N}_1^n$.
—

We define

$$\begin{aligned}
\text{value} &: \mathcal{V} \to \Omega \\
\text{value}(a) &:= \{a\} \\
\text{value} \begin{pmatrix} C_1\ T_{1,1} \ldots\ T_{1,k(1)} \\ |\ \ldots \\ |\ C_n\ T_{n,1} \ldots\ T_{n,k(n)} \end{pmatrix} &:= \\
&\{C_i\ t_{i,1} \ldots\ t_{i,k(1)} \mid \forall i \in \mathbb{N}_1^n, j \in \mathbb{N}_1^{k(i)}.t_{i,j} \in \text{value}(T_{i,j})\} \\
\text{value}(\{l_1 : T_1, \ldots, l_n : T_n\}) &:= \{\{l_1 = t_1, \ldots, l_n = t_n\} \mid \forall i \in \mathbb{N}_1^n.t_i \in \text{value}(T_i)\} \\
\text{value}(T_1 \to T_2) &:= \{f \mid f : \text{value}(T_1) \to \text{value}(T_2)\} \\
\text{value}(\forall a.T) &:= \text{value}(T)
\end{aligned}$$

We write $v : T$ to express that v lives in value(T). Note that $v : T$ is not a predicate.

For any terminal C_i we also define the function

$$\begin{aligned}\text{cons}_{C_i} &: T_{i,1} \to \cdots \to T_{i,k_1} \to T \\ \text{cons}_{C_i}(t_1, \ldots, t_n) &:= C_i\ t_1 \ldots\ t_n\end{aligned}$$

called a *constructor* of T.

For free type variables the corresponding set of values may change.

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.