3 Formal definition of Elm

First, we define some notations:

- $\mathbb{N}$ is the set of the natural numbers starting from 1.
- $\mathbb{N}_0$ is the set of the natural numbers starting from 0.
- $\mathbb{N}_a^b := \{i \in \mathbb{N}_0 | a \leq i \wedge i \leq b\}$ is the set of the natural numbers between a and b.
- We will use "." to separate a quantifier from a statement: $\forall a.F$ and $\exists a.F$, where a is a variable and F is a formula.
- Function types will be written as $a_1 \to \cdots \to a_n \to b$ instead of $a_1 \times \cdots \times a_n \to b$, thus an n-ary function is represented as a unary function whoes result is a (n-1)-ary function. This concept is called "currying".
- We allow the use of lambda notation for functions, i.e. $\lambda x.T$ denotes the function f defined by the equation $f(x) = T$ where T is a term.

3.1 Hindley-Milner Type System

For functional languages considered in this thesis we will use a Hindley-Milner type system [DM82]. The main idea 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. In the following, we give a formal definition of this type system.

3.1.1 Notion of Types

We will first introduce types, afterwards we will define how types relate to sets by explicitly defining the values of types as finite sets. Types are split in *mono types* and *poly types*. Mono types can contain so called *type variables* that can then be bound by a quantifier within 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 define

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

by using the following predicates:

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

Note that the quantifier μC is called a *recursive quantifier*. By using the symbol C we can describe a recursive structure in a non recursive way. That said we need to ensure that every algebraic type has a non-recursive case (called a base case). That's why we require $\exists i \in \mathbb{N}. \forall j \in \mathbb{N}_1^{k(i)}. T_{i,j} \neq C$.

Axiom 3.1

We consider the types T_i for $i \in \mathbb{N}$ in a product type as unordered, i.e.,

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

for all symbols $a, b, \dots$ and mono types $T_1, T_2, \dots$.

Example 3.1

The symbol `Char` is a type variable. The expression `Sequence Char` is a type application. These expressions 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

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

Note that we use the symbol _ to specify a symbol that i only used once in the definition. Multiple occurrences of _ would be seen as multiple difference symbols. We call _ a *wild card*.

Example 3.3

$List = \forall a.\mu C.Empty \mid Cons\ a\ C$ is a poly type whose body $\mu C.Empty \mid Cons\ a\ C$ is an algebraic 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 $T = \mu C.C_1\ T_{1,1} \dots T_{1,k_1} \mid \dots \mid C_n\ T_{n,1} \dots T_{n,k_n}$ be a algebraic type.
—

We call

- C_i a *terminal* of T and
- $C_i\ T_{i,1} \dots T_{i,k_i}$ a *sort* of T for all instantiation of all type-variables in $T_{i,j}$ by mono types that do not contain type variables.

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.

$$Int ::= \mu\_.0 \mid Pos\ Nat \mid Neg\ Nat$$

In this case numbers like 1, $Succ\ 1$ for Nat or $Neg\ (Succ\ 1)$ for Int are sorts, whereas 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 is a *label* 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 $T :\Leftrightarrow a \in \mathit{free}(T)$
- a is *bound* in $T :\Leftrightarrow a \notin \mathit{free}(T)$ and a occurs in T.

where

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

Definition 3.5: Partial function

Let T_1 and T_2 be sets and $f \subseteq T_1 \times T_2$.
—

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

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

A type can be substituted by replacing a bound 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, $k \in \mathbb{N}_1^n \to \mathbb{N}_0$ 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

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

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

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

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

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

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 $\sqsubseteq$ on poly types as

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

From a given type T_1 we can construct a more specialized type T_2 such that $T_1 \sqsubseteq T_2$ by the following steps

1. First replace all bounded variables a_i with types S_i.
2. Next rebound any new variables b_i 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 $\forall a.C\ a$, $\forall a.\forall b.a \to b$ and so on. The leaves of the tree are all possible mono types.

3.1.2 Interpretation of Types

Before we interpret a type, we will first introduce a set of labelled 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

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

Note that values of an ordered product type are equivalent to tuples:

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

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

Definition 3.10: Application Constructor

Let $n \in \mathbb{N}_0$. Let T be a mono type. Let $\{a_1, \ldots, a_n\} := \text{free}(T)$.

—

We call the function

$$(\forall a_1 \ldots a_n . T) : \underbrace{\mathcal{T} \to \cdots \to \mathcal{T}}_{n \text{ times}} \to \mathcal{T}$$
$$(\forall a_1 \ldots a_n . T)(T_1, \ldots, T_n) := [\forall a_1 \ldots a_n . T]_{\{(a_1, T_1), \ldots, (a_n, T_n)\}}.$$

the *application constructor* of T.

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

Definition 3.11: Context

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

Note that mono types with no free variables are considered to be application constructors with no arguments.

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

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

using the following helper function.

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:

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

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

The base case of this recursive function is in $\text{rvalues}_\Gamma(0, T)$ for a given T.

As an example we will prove that the values of Nat from example 3.5 are isomorphic to the natural numbers.

Theorem 3.1

Let the algebraic type Nat defined as $Nat := \mu C.1|Succ\ C$.
—

Then we have:

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

Proof. Let Γ be a context.

First we will introduce a new notation in order to simplify the proof: For any $n \in \mathbb{N}_0$ We define $Succ^n\ 1 := \underbrace{Succ\ \ldots\ Succ}_{n \text{ times}}\ 1$. Note that $Succ^0\ 1 = 1$.

Next we show by induction over $n \in \mathbb{N}_0$ that

$$\text{rvalues}_\Gamma(n, \mu C.1|Succ\ C) = \{Succ^i\ 1 | i \in \mathbb{N}_0^n\}. \tag{1}$$

Base case: $\text{rvalues}_\Gamma(0, \mu C.1|Succ\ C) = \{1\} = \{Succ^0\}$. This is true.
Inductive step:
Assuming $\text{rvalues}_\Gamma(n, \mu C.1|Succ\ C) = \{Succ^i\ 1 | i \in \mathbb{N}_0^n\}$, we will prove $\text{rvalues}_\Gamma(n+1, \mu C.1|Succ\ C) = \{Succ^i\ 1 | i \in \mathbb{N}_0^{n+1}\}$.

$$
\begin{aligned}
&\text{rvalues}_\Gamma(n+1, \mu C.1|Succ\ C) \\
&= \left\{ C_i\ v_1 \dots v_n \;\middle|\; \begin{array}{l} i \in \mathbb{N}_1^n \\ \wedge \forall j \in \mathbb{N}_1^{k(i)}.v_j \in \begin{cases} \text{rvalues}_\Gamma(n, T) & \text{if } T_{i,j} = C \\ \text{values}_\Gamma(T_{i,j}) & \text{else} \end{cases} \end{array} \right\} \\
&= \left\{ C_i\ v_1 \dots v_n \;\middle|\; \begin{array}{l} i \in \mathbb{N}_1^n \\ \wedge \forall j \in \mathbb{N}_1^{k(i)}.v_j \in \begin{cases} \{Succ^k\ 1 | k \in \mathbb{N}_0^n\} & \text{if } T_{i,j} = C \\ \text{values}_\Gamma(T_{i,j}) & \text{else} \end{cases} \end{array} \right\} \\
&= \{1\} \cup \{Succv | v \in \{Succ^i\ 1 | i \in \mathbb{N}_0^n\}\} \\
&= \{Succ^i\ 1 | i \in \mathbb{N}_0^{n+1}\}
\end{aligned}
$$

Now we will prove

$$\text{values}(\mu C.1|Succ\ C) = \{Succ^n\ 1 | n \in \mathbb{N}_0\}. \tag{2}$$

"$\subseteq$": Let

$$x \in \text{values}(\mu C.1|Succ\ C) = \bigcup_{i \in \mathbb{N}_0} \text{rvalues}_\Gamma(i, \mu C.1|Succ\ C).$$

This means, there exists an $i \in \mathbb{N}_0$ such that

$$x \in \text{rvalues}_\Gamma(i, \mu C.1|Succ\ C) \overset{(1)}{=} \{Succ^k\ 1 | k \in \mathbb{N}_0^i\}.$$

Therefore there exists a $k \in \mathbb{N}_0^i$ in dependence of said i, such that $x = Succ^k\ 1$. Thus, in conclusion, $x \in \{Succ^n\ 1 | n \in \mathbb{N}_0\}$.

"$\supseteq$": Let $x \in \{Succ^n\ 1 | n \in \mathbb{N}_0\}$. We have to show that

$$x \in \text{values}(\mu C.1|Succ\ C) = \bigcup_{n \in \mathbb{N}_0} \text{rvalues}_\Gamma(n, \mu C.1|Succ\ C).$$

From the assumption we know that there exists a $n \in \mathbb{N}_0$ such that $x = Succ^n\ 1$. Therefore, we can infer

$$
\begin{aligned}
Succ^n\ 1 \in \{Succ^i\ 1 | i \in \mathbb{N}_0^n\} &\overset{(1)}{=} \text{rvalues}_\Gamma(n, \mu C.1|Succ\ C) \\
&\subseteq \bigcup_{n \in \mathbb{N}_0} \text{rvalues}_\Gamma(n, \mu C.1|Succ\ C)
\end{aligned}
$$

and are done.

To summarize, we have just shown that

$$\text{values}(Nat) = \text{values}(\mu C.1|Succ\ C) \overset{(2)}{=} \{Succ^n\ 1 | n \in \mathbb{N}_0\}.$$

For the last step, we define a well-order on $\{Succ^n\ 1 | n \in \mathbb{N}_0\}$:

$$\forall n, m \in \mathbb{N}_0. Succ^n < Succ^m :\Leftrightarrow n < m$$

The order is defined over the order on the natural numbers and thus a well-order. We know that a set with a well-order is isomorph to the natural numbers and thus, $\{Succ^n\ 1 | n \in \mathbb{N}_0\} \cong \mathbb{N}$. □

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.