3.4. Denotational Semantic

We will now expore the semantics of the formal language. To do so, we first define a new context.

Definition 3.1: Variable Context

Let Γ be a type context.

—

$\Delta : \mathcal{V} \nrightarrow \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(T)$ is called a *variable context*.

The semantics of the type signature was already defined in the last section, as the semantic of a type signature is its type. We therefore define the same concept but now in a denotational style.

Definition 3.2: Type Signature Semantic

Let $T, T' \in \mathcal{T}$, $c, a_0, a \in \mathcal{V}$. Let $t_0, t_1, t_2 \in$ `<type>`, $ltf \in$ `<list-type-fields>` and $lt \in$ `<list-type>`. Let Γ be a type context.

—

Let $s \in (\mathcal{V} \times \mathcal{T})^*$ for the following function.

$$
\begin{aligned}
[\![.]\!]_\Gamma : &\texttt{<list-type-fields>} \to (\mathcal{V} \times \mathcal{T})^* \\
[\![\texttt{""}]\!]_\Gamma = s :\Leftrightarrow & s = (\,) \\
[\![a_0 \quad \texttt{":"} \quad t_0 \quad \texttt{","} \quad ltf]\!]_\Gamma = s :\Leftrightarrow & T_0 = [\![t_0]\!]_\Gamma \\
& \wedge\, s = ((a_0, T_0), \ldots, (a_n, T_n)) \\
& \wedge\, [\![ltf]\!]_\Gamma = ((a_1, T_1), \ldots, (a_n, T_n)) \\
& \text{where } n \in \mathbb{N} \text{ and } T_i \in \mathcal{T}, a_i \in \mathcal{V} \text{ for all } i \in \mathbb{N}_0^n
\end{aligned}
$$

Let $s \in \mathcal{T}^*$ for the following function.

$$
\begin{aligned}
[\![.]\!]_\Gamma : &\texttt{<list-type>} \to \mathcal{T}^* \\
[\![\texttt{""}]\!]_\Gamma = s :\Leftrightarrow & s = (\,) \\
[\![t_0 \;\; lt]\!]_\Gamma = s :\Leftrightarrow & T_0 = [\![t_0]\!]_\Gamma \\
& \wedge\, [\![lt]\!]_\Gamma = (T_1, \ldots, T_n) \\
& \wedge\, s = (T_0, \ldots, T_n) \\
& \text{where } n \in \mathbb{N} \text{ and } T_i \in \mathcal{T} \text{ for all } i \in \mathbb{N}_0^n
\end{aligned}
$$

Let $s \in \mathcal{T}$ for the following function.

$$
\begin{aligned}
[\![.]\!]_\Gamma &: \texttt{<type>} \to \mathcal{T} \\
[\![\texttt{"Bool"}]\!]_\Gamma = s &:\Leftrightarrow s = \mathit{Bool} \\
[\![\texttt{"Int"}]\!]_\Gamma = s &:\Leftrightarrow s = \mathit{Int} \\
[\![\texttt{"List"}\ \ t]\!]_\Gamma = s &:\Leftrightarrow T = [\![\texttt{t}]\!]_\Gamma \wedge s = \mathit{List}\ T \\
&\quad \text{where } T \in \mathcal{T} \\
[\![\texttt{"("}\ \ t_1\ \texttt{","}\ \ t_2\ \ \texttt{")"}]\!]_\Gamma = s &:\Leftrightarrow T_1 = [\![t_1]\!]_\Gamma \wedge T_2 = [\![t_2]\!]_\Gamma \wedge s = (T_1, T_2) \\
&\quad \text{where } T_1, T_2 \in \mathcal{T} \\
[\![\texttt{"\{"}\ \ \mathit{ltf}\ \ \texttt{"\}"}]\!]_\Gamma = s &:\Leftrightarrow [\![\mathit{ltf}]\!]_\Gamma = ((a_1, T_1), \ldots, (a_n, T_n)) \\
&\quad \wedge s = \{a_1 : T_1, \ldots, a_n : T_n\} \\
&\quad \text{where } n \in \mathbb{N} \text{ and } T_i \in \mathcal{T}, a_i \in \mathcal{V} \text{ for all } i \in \mathbb{N}_0^n \\
[\![t_1\ \ \texttt{"->"}\ \ t_2]\!]_\Gamma = s &:\Leftrightarrow [\![t_1]\!]_\Gamma = T_1 \wedge [\![t_2]\!]_\Gamma = T_2 \wedge s = T_1 \to T_2 \\
[\![c\ \ \mathit{lt}]\!]_\Gamma = s &:\Leftrightarrow (c, T) \in \Gamma \\
&\quad \wedge (T_1, \ldots, T_n) = [\![\mathit{lt}]\!]_\Gamma \\
&\quad \wedge T' = \overline{T}\ T_1 \ldots T_n \\
&\quad \wedge s = T' \\
&\quad \text{where } n \in \mathbb{N}, T, T' \in \mathcal{T} \text{ and } T_i \in \mathcal{T} \text{ for all } i \in \mathbb{N}_1^n \\
[\![a]\!]_\Gamma = s &:\Leftrightarrow s = \forall b.b
\end{aligned}
$$

The semantics for <type> are equivalent to the corresponding inference rules. While both directions are valid, we will only need to show that the judgment $\Gamma \vdash t : T$ ensures $[\![\texttt{t}]\!]_\Gamma = T$.

Theorem 3.1

Let Γ be type context, $t \in$ <type> and $T \in \mathcal{T}$.
—
The Judgement $\Gamma \vdash t : T$ can be derived if $[\![t]\!]_\Gamma = T$.

Proof. Let Γ be a type context, $t \in$ <type> and $T \in \mathcal{T}$.

- **case** $t =$ `"Bool"`: Then $\Gamma \vdash t : \mathit{Bool}$ and $[\![t]\!]_\Gamma = \mathit{Bool}$.
- **case** $t =$ `"Int"`: Then $\Gamma \vdash t : \mathit{Int}$ and $[\![t]\!]_\Gamma = \mathit{Int}$.
- **case** $t =$ `"List"` t_2, for $t_2 \in$ <type>: By induction hypothesis, assume $\Gamma \vdash t_2 : T_2$ and $[\![t_2]\!]_\Gamma = T_2$ for given $T_2 \in \mathcal{T}$. Then $\Gamma \vdash t : \mathit{List}\ T_2$ and $[\![t]\!]_\Gamma = \mathit{List}\ T_2$.
- **case** $t =$ `"("` t_1 `","` t_2 `")"`, for $t_1, t_2 \in$ <type>: By induction hypothesis, assume $\Gamma \vdash t_1 : T_1$ and $\Gamma \vdash t_2 : T_2$ for given $T_1, T_2 \in \mathcal{T}$. Then $\Gamma \vdash t : (T_1, T_2)$ and $[\![t]\!]_\Gamma = (T_1, T_2)$.
- **case** $t =$ `"{"` ltf `"}"`, for $\mathit{ltf} \in$ <list-type-field>: By induction over <list-type-field>, assume $\mathit{ltf} =$ `""`, then $\Gamma \vdash \mathit{ltf} : \{\ \}$ and $[\![\mathit{lft}]\!]_\Gamma = \{\ \}$. Next, assume for given $\mathit{ltf}_1 \in$ <list-type-field> that

$\Gamma \vdash ltf_1 : \{a_1 : T_1, \ldots, a_n : T_n\}$ and $[\![ltf_1]\!]_\Gamma = \{a_1 : T_1, \ldots, a_n : T_n\}$ for given $T_i \in \mathcal{T}, a_i \in \mathcal{V}$ and $i \in \mathbb{N}_1^n, n \in \mathbb{N}_0$. Let $lft = a_0$ `":"` T_0 `","` lft_1 for given $a_0 \in \mathcal{V}$ and $T_0 \in \mathcal{T}$. Then $\Gamma \vdash lft : \{a_0 : T_0, a_1 : T_1 \ldots a_n : T_n\}$ and $[\![lft]\!]_\Gamma = \{a_0 : T_0, a_1 : T_1 \ldots a_n : T_n\}$. We can therefore conclude that for $\Gamma \vdash ltf : T$ for inductively derived $T \in \mathcal{T}$ that $\Gamma \vdash t : T$ and $[\![t]\!]_\Gamma = T$.

- **case** $t = t_1$ `"->"` t_2, for $t_1, t_2 \in$ `<type>`: By induction hypothesis, assume $\Gamma \vdash t_i : T_i$ and $[\![t_i]\!]_\Gamma = T_i$ for $i \in \{1, 2\}$ and given $T_1, T_2 \in \mathcal{T}$. Then $\Gamma \vdash t : T_1 \to T_2$ and $[\![t]\!]_\Gamma = T_1 \to T_2$.
- **case** $t = c\ l$ for $l \in$ `<list-type>` and $c \in$ `<upper-var>` with $(c, T') \in \Gamma$ with $T' \in \mathcal{T}$: By induction over `<list-type>`, assume $l =$ `""`, then $\Gamma \vdash l : ()$ and $[\![l]\!]_\Gamma = ()$. Next, assume for given $l_1 \in$ `<list-type>` that $\Gamma \vdash l_1 : (T_1, \ldots, T_n)$ for $T_i \in \mathcal{T}$,$i \in \mathbb{N}_1^n$ and $n \in \mathbb{N}_0$. Let $l =\ t_0\ l_1$ for $t_0 \in$ `<type>` such that $\Gamma \vdash t_0 : T_0$ and $[\![t_0]\!]_\Gamma = T_0$ and $T_0 \in \mathcal{T}$, then $\Gamma \vdash l : (T_0, T_1, \ldots, T_n)$. We can therefore assume $\Gamma \vdash l : (T_0, \ldots, T_n)$ and $[\![l]\!]_\Gamma = (T_1, \ldots, T_n)$ for $T_i \in \mathcal{T}$,$i \in \mathbb{N}^n$ and $n \in \mathbb{N}_0$. Then $\Gamma \vdash t : \overline{T'}\ T_1\ \ldots\ T_n$ and $[\![t]\!]_\Gamma = \overline{T'}\ T_1\ \ldots\ T_n$, where $\overline{T'}$ represents the application constructor of T'
- **case** $t = a$ for $a \in \mathcal{V}$ then $\Gamma \vdash t : \forall\ b.b$ and $[\![t]\!]_\Gamma = \forall\ b.b$

□

We have already seen the pattern matching predicate in the last section. It is now time to actually define its meaning.

Definition 3.3: Pattern Semantic

Let Γ be a type context and let $\Theta, \Theta_1, \Theta_2, \Theta_3$ be variable contexts. Let $p, p_1, p_2 \in$ `<pattern>`, $lpl \in$ `<list-pattern-list>`, $lps \in$ `<list-pattern-sort>` and $lpv \in$ `<list-pattern-vars>`. Let $b \in$ `<bool>` and $i \in$ `<int>`. Let $c, a \in \mathcal{V}$.

—

Let $s \in \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(List\ T))$ for the following predicate.

$$
\begin{aligned}
\text{match}_\Theta :& (\bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(List\ T)) \times \texttt{<list-pattern-list>} \\
\text{match}_\Theta(s, \texttt{""}) :\Leftrightarrow& [\,] = s \\
\text{match}_{\Theta_3}(s, p\ \texttt{","}\ lpl) :\Leftrightarrow& [a_0, \ldots, a_n] = s \\
& \wedge \text{match}_{\Theta_1}(a_0, p) \wedge \text{match}_{\Theta_2}(a_1, \ldots, a_n, lpl) \\
& \wedge \varnothing = \Theta_1 \cap \Theta_2 \wedge \Theta_3 = \Theta_1 \cup \Theta_2 \\
& \text{where } n \in \mathbb{N} \text{ and } a_i \in \mathcal{V} \text{ for all } i \in \mathbb{N}_0^n.
\end{aligned}
$$

Let $s \in \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(T)^*$ for the following predicate.

$$\begin{aligned}
\text{match}_\Theta : \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(T)^* \times \texttt{<list-pattern-sort>} \\
\text{match}_\Theta(s, \texttt{""}) :\Leftrightarrow & (\ ) = s \\
\text{match}_{\Theta_3}(s, p\ lps) :\Leftrightarrow & (s_1, \ldots, s_n) = s \\
& \wedge \text{match}_{\Theta_1}(s_1, p) \wedge \text{match}_{\Theta_2}(s_2, \ldots, s_n, lps) \\
& \wedge \varnothing = \Theta_1 \cap \Theta_2 \wedge \Theta_3 = \Theta_1 \cup \Theta_2 \\
& \text{where } n \in \mathbb{N} \text{ and } s_i \in \text{value}(T) \text{ with } T \in \mathcal{T} \text{ for all } i \in \mathbb{N}_0^n.
\end{aligned}$$

Let $s \in \mathcal{V}^*$ for the following function.

$$\begin{aligned}
[\![.]\!] : & \texttt{<list-pattern-vars>} \to \mathcal{V}^* \\
[\![\texttt{""}]\!] = s :\Leftrightarrow & s = (\ ) \\
[\![a_0\ \ lpv]\!] = s :\Leftrightarrow & (a_1, \ldots, a_n) = [\![lpv]\!] \wedge (a_0, \ldots, a_n) = s \\
& \text{where } n \in \mathbb{N} \text{ and } a_i \in \mathcal{V} \text{ for all } i \in \mathbb{N}_0^n.
\end{aligned}$$

Let $s \in \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(T)$ for the following predicate.

$$
\begin{aligned}
\text{match}_\Theta : & \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(T) \times \texttt{<pattern>} \\
\text{match}_\Theta(s, b) :\Leftrightarrow & \wedge b \in \texttt{<bool>} \wedge s = [\![b]\!]_{\Gamma,\varnothing} \\
\text{match}_\Theta(s, i) :\Leftrightarrow & \wedge i \in \texttt{<int>} \wedge s = [\![b]\!]_{\Gamma,\varnothing} \\
\text{match}_\Theta(s, \texttt{"[" } lpl \texttt{ "]"}) :\Leftrightarrow & \text{match}_\Theta(s, lpl) \\
\text{match}_{\Theta_3}(s, \texttt{"(" } p_1 \texttt{ "," } p_2 \texttt{ ")"}) :\Leftrightarrow & (s_1, s_2) = s \\
& \wedge \text{match}_{\Theta_1}(s_1, p_1) \wedge \text{match}_{\Theta_2}(s_2, p_2) \\
& \wedge \varnothing = \Theta_1 \cap \Theta_2 \wedge \Theta_3 = \Theta_1 \cup \Theta_2 \\
& \text{where } s_1 \in \text{value}(T_1), s_2 \in \text{value}(T_2) \text{ for} \\
& T_1, T_2 \in \mathcal{T} \\
\text{match}_\Theta(s, c\ lps) :\Leftrightarrow & c\ s_1\ \ldots\ s_n = s \wedge \text{match}_\Theta((s_1, \ldots, s_n), lps) \\
& \text{where } n \in \mathbb{N} \text{ and } s_i \in \text{value}(T) \text{ with } T \in \mathcal{T} \text{ for} \\
& \text{all } i \in \mathbb{N}_0^n. \\
\text{match}_\Theta(s, a) :\Leftrightarrow & s \in \mathcal{V} \wedge \Theta = \{(a, s)\} \\
\text{match}_{\Theta_2}(s, p \texttt{ "as" } a) :\Leftrightarrow & \text{match}_{\Theta_1}(s, p) \\
& \wedge \varnothing = \Theta_1 \cap \{(a, s)\} \wedge \Theta_2 = \Theta_1 \cup \{(a, s)\} \\
\text{match}_\Theta(s, \texttt{"\{" } lpv \texttt{ "\}"}) :\Leftrightarrow & (a_1, \ldots, a_n) = [\![lpv]\!] \\
& \wedge \{a_1 = e_1, \ldots, a_n = e_n\} = s \\
& \wedge \Theta = \{(a_1, e_1), \ldots, (a_n, e_n)\} \\
& \text{where } n \in \mathbb{N} \text{ and } a_i \in \mathcal{V} \text{ for all } i \in \mathbb{N}_0^n. \\
\text{match}_{\Theta_3}(s, p_1 \texttt{ "::" } p_2) :\Leftrightarrow & (s_1, \ldots, s_n) = s \wedge \text{match}_{\Theta_1}(s_1, p_1) \\
& \wedge \text{match}_{\Theta_2}((s_2, \ldots, s_n), p_2) \\
& \wedge \varnothing = \Theta_1 \cap \Theta_2 \wedge \Theta_3 = \Theta_1 \cup \Theta_2 \\
& \text{where } n \in \mathbb{N} \text{ and } s_i \in \text{value}(T) \text{ with } T \in \mathcal{T} \text{ for} \\
& \text{all } i \in \mathbb{N}_0^n. \\
\text{match}_\Theta(\texttt{"\_"}) :\Leftrightarrow & \varnothing = \Theta
\end{aligned}
$$

An Elm program is nothing more than an expression. Semantics of an expression is therefore the heart piece of this section.

Definition 3.4: Expression Semantic

Let Γ be a type context and let Δ, Θ be variable contexts. Let $a, a_0, a_1 \in \mathcal{V}$, $e, e_1, e_2, e_3 \in$ `<exp>`. Let $lef \in$ `<list-exp-field>`, $t \in$ `<type>`, $p \in$ `<pattern>`, $lc \in$ `<list-case>`, $b \in$ `<bool>`, $nr \in \mathbb{N}$, $le \in$ `<list-exp>` and $mes \in$ `<maybe-expression-sign>`.

—

Let $s \in (\mathcal{V} \times \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(T))^*$ for the following function.

$$[\![.]\!]_{\Gamma,\Delta} : \texttt{<list-exp-field>} \to (\mathcal{V} \times \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(T))^*$$

$$[\![a \quad \texttt{"="} \quad e]\!]_{\Gamma,\Delta} = s_1 :\Leftrightarrow s_2 = [\![e]\!]_{\Gamma,\Delta} \wedge ((a, s_2)) = s_1$$
$$\text{where } s_2 \in \text{value}_\Gamma(T) \text{ for } T \in \mathcal{T}$$

$$[\![a_1 \quad \texttt{"="} \quad e \quad \texttt{","} \quad lef]\!]_{\Gamma,\Delta} = s_3 :\Leftrightarrow ((a_1, s_1)) = [\![a \quad \texttt{"="} \quad e]\!]_{\Gamma,\Delta}$$
$$\wedge ((a_2, s_2), \ldots, (a_n, s_n)) = [\![lef]\!]_{\Gamma,\Delta}$$
$$\wedge ((a_1, s_1), \ldots, (a_n, s_n)) = s_3$$
$$\text{where } n \in \mathbb{N} \text{ and } a_i \in \mathcal{V}, s_i \in \text{value}_\Gamma(T_i)$$
$$\text{for } T_i \in \mathcal{T} \text{ for } i \in \mathbb{N}_0^n$$

$$[\![.]\!]_{\Gamma,\Delta} : \texttt{<maybe-exp-sign>} \to (\,)$$
$$[\![\texttt{""}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow (\,) = s$$
$$[\![a \quad \texttt{":"} \; t \quad \texttt{";"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow (\,) = s$$

Let $s \in \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(T))$ for the following function.

$$[\![.]\!]_{\Gamma,\Delta} : \texttt{<exp>} \to \texttt{<list-case>} \to \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(T))$$

$$[\![e_1, p \quad \texttt{"->"} \quad e_2]\!]_{\Gamma,\Delta} = s :\Leftrightarrow \text{match}_\Theta(e_1, p) \wedge [\![e_2]\!]_{\Gamma,\Delta\cup\Theta} = s$$

$$[\![e_1, p \quad \texttt{"->"} \quad e_2 \quad \texttt{";"} \quad lc]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s = \begin{cases} [\![e_2]\!]_{\Gamma,\Delta\cup\Theta} & \text{if match}_\Theta(e_1, p) \\ [\![e_1, lc]\!]_{\Gamma,\Delta} & \text{else} \end{cases}$$

Let $s \in \text{value}_\varnothing(Bool)$ for the following function.

$$[\![.]\!] : \texttt{<Bool>} \to \text{value}_\varnothing(Bool)$$
$$[\![b]\!] = s :\Leftrightarrow \begin{cases} True & \text{if } b = \texttt{"True"} \\ False & \text{if } b = \texttt{"False"} \end{cases}$$

Let $s \in \text{value}_\varnothing(Int)$ for the following function.

$$[\![.]\!] : \texttt{<int>} \to \text{value}_\varnothing(Int)$$
$$[\![\texttt{"0"}]\!] = s :\Leftrightarrow 0 = s$$
$$[\![\texttt{"-"} \quad nr]\!] = s :\Leftrightarrow Neg\ Succ^{nr}\ 1$$
$$[\![nr]\!] = s :\Leftrightarrow Pos\ Succ^{nr}\ 1$$

Let $s \in \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(T))^*$ for the following function.

$$[\![.]\!]_{\Gamma,\Delta} : \texttt{<list-exp>} \to \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(T))^*$$

$$[\![\texttt{""}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow (\,) = s$$

$$[\![e \quad \texttt{","} \quad le]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s_1 = [\![e]\!]_{\Gamma,\Delta} \wedge (s_2, \ldots, s_n) = [\![le]\!]_{\Gamma,\Delta} \wedge (s_1, \ldots, s_n) = s$$
$$\text{where } n \in \mathbb{N} \text{ and } s_i \in \text{value}_\Gamma(T_i), T_i \in \mathcal{T} \text{ for each } i \in \mathbb{N}_0^n$$

Let $s \in \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(T)$ for the following function.

$$[\![.]\!]_{\Gamma,\Delta} : \texttt{<exp>} \to \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(T)$$

$$[\![\texttt{"foldl"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s = \lambda f.\lambda e_1.\lambda l_1.$$
$$\begin{cases} e_1 & \text{if } [\,] = l_1 \\ f(e_2, s(f, e_1, l_2)) & \text{if } \mathit{Cons}\ e_2\ l_2 = l_1 \end{cases}$$
$$\text{where } e_1 \in \text{value}_\Gamma(T_1), e_2 \in \text{value}_\Gamma(T_2)$$
$$\text{and } l_1, l_2 \in \text{value}_\Gamma(\mathit{List}\ T_2) \text{ and}$$
$$f \in \text{value}_\Gamma(T_2 \to T_1 \to T_1) \text{ for } T_1, T_2 \in \mathcal{T}$$

$$[\![\texttt{"(::)"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s = \lambda e.\lambda l.\mathit{Cons}\ e\ l$$
$$\text{where } e \in \text{value}_\Gamma(T) \text{ and } l \in \text{value}_\Gamma(\mathit{List}\ T)$$
$$\text{for } T \in \mathcal{T}$$

$$[\![\texttt{"(+)"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s = \lambda n.\lambda m.n + m$$
$$\text{where } n, m \in \mathbb{N}$$

$$[\![\texttt{"(-)"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s = \lambda n.\lambda m.n - m$$
$$\text{where } n, m \in \mathbb{N}$$

$$[\![\texttt{"(*)"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s = \lambda n.\lambda m.n * m$$
$$\text{where } n, m \in \mathbb{N}$$

$$[\![\texttt{"(//)"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow \begin{cases} s = \lambda n.\lambda m.\left\lfloor \frac{n}{m} \right\rfloor & \text{if } m \neq 0 \\ 0 & \text{else} \end{cases}$$
$$\text{where } n, m \in \mathbb{N}$$

$$[\![\texttt{"(<)"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s = \lambda n.\lambda m.n < m$$
$$\text{where } n, m \in \mathbb{N}$$

$$[\![\texttt{"(==)"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s = \lambda n.\lambda m.(n = m)$$
$$\text{where } n, m \in \mathbb{N}$$

$$[\![\texttt{"not"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s = \lambda b.\neg b$$
$$\text{where } b \in \text{value}_\Gamma(\mathit{Bool})$$

$$[\![\texttt{"(\&\&)"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s = \lambda b_1.\lambda b_2.b_1 \wedge b_2$$
$$\text{where } b_1, n_2 \in \text{value}_\Gamma(Bool)$$

$$[\![\texttt{"(||)"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s = \lambda b_1.\lambda b_2.b_1 \vee b_2$$
$$\text{where } b_1, n_2 \in \text{value}_\Gamma(Bool)$$

$$[\![e_1 \quad \texttt{"|>"} \quad e_2]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s' = [\![e_1]\!]_{\Gamma,\Delta} \wedge f = [\![e_2]\!]_{\Gamma,\Delta} \wedge f(s_1) = s$$
$$\text{where } f \in \text{value}(T_1 \to T_2), s' \in \text{value}(T_1) \text{ for } T_1, T_2 \in \mathcal{T}$$

$$[\![e_1 \quad \texttt{">>"} \quad e_2]\!]_{\Gamma,\Delta} = s :\Leftrightarrow g = [\![e_1]\!]_{\Gamma,\Delta} \wedge f = [\![e_2]\!]_{\Gamma,\Delta} \wedge f \circ g = s$$
$$\text{where } g \in \text{value}(T_1 \to T_2), f \in \text{value}(T_2 \to T_3) \text{ for } T_1, T_2, T_2 \in \mathcal{T}$$

$$\left[\!\!\left[\begin{matrix}\texttt{"if"}\ e_1\ \texttt{"then"} \\ e_2\ \texttt{"else"}\ e_3\end{matrix}\right]\!\!\right]_{\Gamma,\Delta} = s :\Leftrightarrow b = [\![e_1]\!]_{\Gamma,\Delta} \wedge s = \begin{cases}[\![e_2]\!]_{\Gamma,\Delta} & \text{if } b \\ [\![e_3]\!]_{\Gamma,\Delta} & \text{if } \neg b\end{cases}$$
$$\text{where } b \in \text{value}(Bool)$$

$$[\![\texttt{"\{"} \quad lef \quad \texttt{"\}"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow ((a_1, s_1), \ldots, (a_n, s_n)) = [\![lef]\!]_{\Gamma,\Delta}$$
$$\wedge \{a_1 = s_1, \ldots, a_n = s_n\} = s$$
$$\text{where } n \in \mathbb{N} \text{ and } a_i \in \mathcal{V}, s_i \in \text{value}(T_i), T_i \in \mathcal{T} \text{ for } i \in \mathbb{N}_0^n$$

$$[\![\texttt{"\{ \}"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow \{\} = s$$

$$[\![\texttt{"\{"} \quad a \quad \texttt{"|"} \quad lef \quad \texttt{"\}"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow ((a_1, s_1), \ldots, (a_n, s_n)) = [\![lef]\!]_{\Gamma,\Delta}$$
$$\wedge (a, \left\{\begin{matrix}a_1 = \_, \ldots, a_n = \_, \\ a_{n+1} = s_{n+1}, \ldots, a_m = s_m\end{matrix}\right\}) \in \Delta$$
$$\wedge \{a_1 = s_1, \ldots, a_m = s_m\} = s$$
$$\text{where } n, m \in \mathbb{N} \text{ such that } n \leq m \text{ and } a_i \in \mathcal{V},$$
$$s_i \in \text{value}(T_i), T_i \in \mathcal{T} \text{ for } i \in \mathbb{N}_0^m$$

$$[\![a_0 \quad \texttt{"."} \quad a_1]\!]_{\Gamma,\Delta} = s :\Leftrightarrow (a_0, \{a_1 : s_1, \ldots\})\Delta \wedge s' = s$$
$$\text{where } s' \in \text{value}(T) \text{for } T \in \mathcal{T}$$

$$\left[\!\!\left[\begin{matrix}\texttt{"let"}\ mes\ a\ \texttt{"="}\ e_1 \\ \texttt{"in"}\ e_2\end{matrix}\right]\!\!\right]_{\Gamma,\Delta} = s :\Leftrightarrow s' = [\![e_1]\!]_{\Gamma,\Delta} \ \wedge [\![e_2]\!]_{\Gamma,\Delta \cup \{(a,s')\}} = s$$
$$\text{where } s' \in \text{value}(T) for T \in \mathcal{T}$$

$$\left[\!\!\left[\begin{matrix}\texttt{"case"}\ e\ \texttt{"of"} \\ \texttt{"["}\ lc\ \texttt{"]"}\end{matrix}\right]\!\!\right]_{\Gamma,\Delta} = s :\Leftrightarrow [\![e, lc]\!]_{\Gamma,\Delta} = s$$

$$[\![e_1 \ e_2]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s_1 = [\![e_1]\!]_{\Gamma,\Delta} \wedge s_2 = [\![e_2]\!]_{\Gamma,\Delta} \wedge s_1(s_2) = s$$
$$\text{where } s_1 \in \text{value}_\Gamma(T_1 \to T_2) \text{ and } s_2 \in \text{value}_\Gamma(T_1) \text{for } T_1, T_2 \in \mathcal{T}$$

$$[\![b]\!]_{\Gamma,\Delta} = s :\Leftrightarrow [\![b]\!] = s$$

$$[\![i]\!]_{\Gamma,\Delta} = s :\Leftrightarrow [\![i]\!] = s$$

$$[\![\texttt{"["}\quad le\quad \texttt{"]"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow (s_1,\ldots,s_n) = [\![le]\!]_{\Gamma,\Delta} \wedge [s_1,\ldots,s_n] = s$$
$$\text{where } n \in \mathbb{N} \text{ and } s_i \in \text{value}_\Gamma(T) \text{ for } T \in \mathcal{T}$$

$$[\![\texttt{"("}\quad e_1\quad \texttt{","}\quad e_2\quad \texttt{")"}]\!]_{\Gamma,\Delta} = s :\Leftrightarrow s_1 = [\![e_1]\!] \wedge s_2 = [\![e_2]\!] \wedge (s_1,s_2) = s$$
$$\text{where } s_1 \in \text{value}_\Gamma(T_1) \text{ and } s_2 \in \text{value}_\Gamma(T_1)$$

$$[\![\texttt{"\textbackslash"}\quad p\quad \texttt{"->"}\quad e]\!]_{\Gamma,\Delta} = s :\Leftrightarrow \text{match}_\Theta(s_1,p) \wedge s_2 = [\![e]\!] \wedge \lambda s_1.s_2 = s$$
$$\text{where } s_1 \in \text{value}_\Gamma(T_1) \text{ and } s_2 \in \text{value}_\Gamma(T_1)$$

$$[\![c]\!]_{\Gamma,\Delta} = s :\Leftrightarrow (c,s) \in \Delta$$

$$[\![a]\!]_{\Gamma,\Delta} = s :\Leftrightarrow (a,s) \in \Delta$$

Statements are, semantically speaking, just functions that either map the type- or variable-context.

Definition 3.5: Statement Semantic

Let Γ be a type context. Let $a, a_0 \in \mathcal{V}$, $t \in$ `<type>`, $lsv \in$ `<list-statement-var>`, $lt \in$ `<list-type>`, $lss \in$ `<list-statement-sort>`, $st \in$ `<statement>`, $ls \in$ `<list-statement>`, $mss \in$ `<maybe-statement-sign>` and $mms \in$ `<maybe-main-sign>`. Let $\mathcal{S}$ be the class of all finite sets.

—

Let $s \in \mathcal{V}^*$ for the following function.

$$[\![.]\!] :\texttt{<list-statement-var>} \to \mathcal{V}^*$$
$$[\![\texttt{""}]\!] = s :\Leftrightarrow (\,) = s$$
$$[\![a_0\quad lsv]\!] = s :\Leftrightarrow (a_1,\ldots,a_n) = [\![lsv]\!] \wedge (a_0,\ldots,a_n) = s$$
$$\text{where } n \in \mathbb{N} \text{ and } a_i \in \mathcal{V} \text{ for } i \in \mathbb{N}_0^n$$

Let $s \in (\mathcal{V} \times \mathcal{T}^*)^*$ for the following function.

$$[\![.]\!]_\Gamma :\texttt{<list-statement-sort>} \to (\mathcal{V} \times \mathcal{T}^*)^*$$
$$[\![a\quad lt]\!]_\Gamma = s :\Leftrightarrow l = [\![lt]\!]_\Gamma \wedge ((a,l)) = s$$
$$\text{where } l \in \mathcal{T}^*$$
$$[\![a_0\quad lt\quad lss]\!]_\Gamma = s :\Leftrightarrow ((a_1,l_1),\ldots,(a_n,l_n)) = [\![lss]\!]$$
$$\wedge\ l_0 = [\![lt]\!]_\Gamma$$
$$\wedge\ ((a_0,l_0),\ldots,(a_n,l_n)) = s$$
$$\text{where } n \in \mathbb{N} \text{ and } a_i \in \mathcal{V}, l_i \in \mathcal{T}^* \text{ for } i \in \mathbb{N}$$

Let $s \in ((\mathcal{V} \nrightarrow \mathcal{T}) \times (\mathcal{V} \nrightarrow \mathcal{S})) \to ((\mathcal{V} \nrightarrow \mathcal{T}) \times (\mathcal{V} \nrightarrow \mathcal{S}))$ for the following function.

$$[\![.]\!] : \texttt{<list-statement>} \to ((\mathcal{V} \nrightarrow \mathcal{T}) \times (\mathcal{V} \nrightarrow \mathcal{S})) \to ((\mathcal{V} \nrightarrow \mathcal{T}) \times (\mathcal{V} \nrightarrow \mathcal{S}))$$

$$[\![\texttt{""}]\!] = s :\Leftrightarrow id = s$$

$$[\![st \quad \texttt{","} \quad ls]\!] = s :\Leftrightarrow f = [\![st]\!] \wedge g = [\![ls]\!] \wedge g \circ f = s$$

where $f, g \in ((\mathcal{V} \nrightarrow \mathcal{T}) \times (\mathcal{V} \nrightarrow \mathcal{S})) \to ((\mathcal{V} \nrightarrow \mathcal{T}) \times (\mathcal{V} \nrightarrow \mathcal{S}))$

$$[\![.]\!] : \texttt{<maybe-statement-sign>} \to (\ )$$

$$[\![\texttt{""}]\!] = s :\Leftrightarrow (\ ) = s$$

$$[\![a \quad \texttt{":"} \quad t \ \texttt{";"}]\!] = s :\Leftrightarrow (\ ) = s$$

Let $s \in ((\mathcal{V} \nrightarrow \mathcal{T}) \times (\mathcal{V} \nrightarrow \mathcal{S})) \to ((\mathcal{V} \nrightarrow \mathcal{T}) \times (\mathcal{V} \nrightarrow \mathcal{S}))$ for the following function.

$$[\![.]\!] : \texttt{<statement>} \to ((\mathcal{V} \nrightarrow \mathcal{T}) \times (\mathcal{V} \nrightarrow \mathcal{S})) \to ((\mathcal{V} \nrightarrow \mathcal{T}) \times (\mathcal{V} \nrightarrow \mathcal{S}))$$

$$[\![mss \quad a \quad \texttt{"="} \quad e]\!](\Gamma, \Delta) = s :\Leftrightarrow s' = [\![e]\!]_{\Gamma,\Delta} \wedge (\Gamma, \Delta \cup \{(a, s')\}) = s$$

where $s' \in \text{value}(T)$ for $T \in \mathcal{T}$

$$\left[\!\!\left[\begin{matrix}\texttt{"type alias"} \\ c \ lsv \ \texttt{"="} \ t\end{matrix}\right]\!\!\right](\Gamma, \Delta) = s :\Leftrightarrow T = [\![\texttt{t}]\!]_\Gamma \wedge (\Gamma \cup \{(c, T)\}, \Delta) = s$$

$$\left[\!\!\left[\begin{matrix}\texttt{"type"} \ c \\ lsv \ \texttt{"="} \ lss\end{matrix}\right]\!\!\right](\Gamma_1, \Delta_2) = s :\Leftrightarrow ((c_1, (T_{1,1}, \ldots, T_{1,k_1})), \ldots, (c_n, (T_{n,1}, \ldots, T_{n,k_n})))$$

$$\wedge\ T_1 = \mu C.c_1 T_{1,1} \ldots T_{1,k_1} | \ldots | c_n T_{n,1} \ldots T_{n,k_n}$$

$$\wedge\ (a_1, \ldots, a_m) = [\![lsv]\!]$$

$$\wedge\ T_2 = \forall a_1. \ldots \forall a_m T_1$$

$$\wedge\ \Gamma_2 = \Gamma_1 \cup \{(c, T_2)\}$$

$$\wedge\ \Delta_2 = \left\{\begin{matrix}(c_1, \lambda t_{1,1}. \ldots, \lambda t_{1,k_1}.c_1 t_{1,1} \ldots t_{1,k_1}) \\ \vdots \\ (c_n, \lambda t_{n,1}. \ldots, \lambda t_{n,k_n}.c_1 t_{n,1} \ldots t_{n,k_n})\end{matrix}\right\}$$

$$\wedge\ \Delta_3 = \Delta_1 \cup \Delta_2$$

$$\wedge\ (\Gamma_2, \Delta_3) = s$$

where $n, m \in \mathbb{N}$ and $k_i \in \mathbb{N}, c_i, a_j \in \mathcal{V}, T_{i,l_i} \in \mathcal{T}$ for $i \in \mathbb{N}_1^n, j \in \mathbb{N}_1^m, l_i \in \mathbb{N}_0^{k_i}$

$$[\![.]\!] : \texttt{<maybe-main-sign>} \to (\ )$$

$$[\![\texttt{""}]\!] = s :\Leftrightarrow (\ ) = s$$

$$[\![\texttt{"main : "} \quad t \quad \texttt{";"}]\!] = s :\Leftrightarrow (\ ) = s$$

Let $s \in \bigcup_{T \in \mathcal{T}} \text{value}_\Gamma(T)$ for the following function.

$$[\![.]\!] : \texttt{<program>} \to \bigcup_{T \in \mathcal{T}} \text{value}_{\varnothing}(T)$$

$$[\![ls \quad mms \quad \texttt{"main = "} \quad e]\!] = s :\Leftrightarrow (\Gamma, \Delta) = [\![ls]\!](\varnothing, \varnothing) \wedge [\![e]\!]_{\Gamma,\Delta} = s$$

where Γ is a type context and Δ is a variable context.

3.5. Soundness

In this section we want to prove the soundness of the inference rules with respect to the semantics. This means we want to ensure that a if we can infer the type of a program, the program has also got a semantic.

This will be done in four steps:

1. Proving the Soundness of the type context.
2. Proving the Soundness of the variable context.
3. Proving the Soundness of expressions for given type and variable contexts.
4. Showing that every program can be rewritten into an expression with an type context and a variable context.

3.5.1. Soundness of the type environment

The type environment is build by the `type alias` statement. To recap, a statement is an operation on on of the evironments. Thus we need to check that the operation described in the inference rule is the same as the operation described in the semantics.

Theorem 3.2

Let Γ_1, Γ_2 be type contexts. Let Δ_1, Δ_2 be variable contexts.

—

The judgment $\Gamma_1, \Delta_1, \texttt{type alias}\ c\ lsv\ \texttt{"="}\ t \vdash \Gamma_2, \Delta_2$ can be derived if,

$$[\![\texttt{"type alias"} \quad t \quad c \quad lsv \quad \texttt{"="} \quad t]\!](\Gamma_1, \Delta_1) = (\Gamma_2, \Delta_2)$$

To do so, we will first proof that given a enviroment

Theorem 3.3

Let $p \in \texttt{<program>}$ and $T \in \mathcal{T}$ such that $p : T$.

—

Then

$$\exists s \in \text{value}_{\varnothing}(T).[\![\texttt{p}]\!] = s$$

Proof. □

A. Appendix: Type System written in K Framework

K Framework[RS14] is a system for designing and formalizing programming languages. It uses rewriting systems that are written in its own K Language.

An K file contains modules with the base module name the same as the k file.

```
require "unification.k"
require "elm-syntax.k"

module ELM-TYPESYSTEM
  imports DOMAINS
  imports ELM-SYNTAX

  configuration <k> $PGM:Exp </k>
                <tenv> .Map </tenv>
  //..

  syntax KResult ::= Type
endmodule
```

You can specify the realm upon which the rewriting system can be executed by using the `configuration` keyword. Here we specify two parts: `<k></k>` containing the expression and `<tenv></tenv>` containing the environment.

We also need to specify the end result using the `KResult` keyword. Once the rewriting system reaches the such an expression, it will stop. If not specified the system might not terminate.

The formal grammar can be translated directly into the K language

```
syntax Type
  ::= "bool"
    | "int"
    | "{}Type"
    | "{" ListTypeFields "}Type" [strict]
    | Type "->" Type       [strict,right]
    | LowerVar
    | "(" Type ")"              [bracket]
    | ..
```

Additionally, we can include meta-information: `strict` to ensure the inner expression get evaluated first, `right/left` in which direction the expressions should be evaluated and `bracket` for brackets.

Rules are written as rewriting rules instead of inference rules.

```
syntax Exp ::= Type
rule E1:Type E2:Type
```

```
  => E1 =Type (E2 -> ?T:Type)
    ~> ?T
syntax KResult ::= Type
```

The rule itself has the syntax `rule . => .`. To ensure the rule gets only executed once all inner expression have been rewritten, we can include an additional `syntax` line before the rule and a `KResult` to ensure that rewriting system keeps on applying rules until a specific result has been reached. Only then it may continue.

Additionally, we have variables starting with an uppercase letter and existentially quantified variables starting with a question mark.

The system itself allows for a more untraditional imperative rewriting system using `~>`. This symbol has only one rule: `rule . ~> A => A` where `.` is the empty symbol. Thus saying, the left part need to be rewritten to `.` before the right part can be changed.

A.1. Implementing Algorithm J

In the original paper by Milner[Mil78] an optimized algorithm is presented for implementing polymorphism in a programming language. This algorithm is imperative but is typically presented as logical rules.

$$\frac{a : T_1 \quad T_2 = inst(T_1)}{\Gamma \vdash_J a : T_2} \qquad \text{[Variable]}$$

$$\frac{\Gamma \vdash_J e_0 : T_0 \quad \Gamma \vdash_J e_1 : T_1 \quad T_2 = newvar \quad unify(T_0, T_1 \to T_2)}{\Gamma \vdash_J e_0 e_1 : T_2} \qquad \text{[Call]}$$

$$\frac{T_1 = newvar \quad \Gamma, x : T_1 \vdash_J e : T_2}{\Gamma \vdash_J \backslash x\texttt{->}e : T_0 \to T_1} \qquad \text{[Lambda]}$$

$$\frac{\Delta_1 \vdash_J e_0 : T_1 \quad \Delta_1, a : \text{insert}_{\Delta_1}(\{T_1\}) \vdash_J e_1 : T_2}{\Delta \vdash_J \texttt{let}x\texttt{=}e_0\texttt{in}e_1 : T_2} \qquad \text{[LetIn]}$$

The imperative functions are *newvar*, *unify* and *inst*. *newvar* creates a new variable, *inst* instantiates a type with new variables and *inify* checks whether two types can be unified.

K Framework has a these imperative functions implemented in the `Unification.k` module. In order to use them, we need to first properly define poly types. This way, we only need to compute the bound variables of a type once.

```
syntax PolyType ::= "forall" Set "." Type
```

Next we tell the system that we want to use the unification algorithm on types.

```
syntax Type ::= MetaVariable
```

Once this is set up, we can use the function `#renameMetaKVariables` for *inst* and `?T` for *newvar*.

```
rule <k> variable X:Id => #renameMetaKVariables(T, Tvs) ...</k>
    <tenv>... X |-> forall Tvs . T
    ...</tenv>

rule <k> fun A:Id -> E:Type => ?T:Type -> E ~> setTenv(TEnv) ...</k>
    <tenv> TEnv:Map => TEnv [ A <- ?T ] </tenv>

syntax KItem ::= setTenv(Map)
  rule <k> T:Type ~> (setTenv(TEnv) => .) ...</k>
   <tenv> _ => TEnv </tenv>
```

Note that the `setTenv` function ensures that ?T is instantiated before its inserted into the environment.

For implementing insert_{Δ} we use `#metaKVariables` for getting all bound variables and `#freezeKVariables` to ensure that variables in the environment need to be newly instantiated when ever they get used.

```
rule <k> let X = T:Type in E => E ~> setTenv(TEnv)
    ...</k>
    <tenv> TEnv
      => TEnv[ X
        <- forall (#metaKVariables(T) -Set #metaKVariables(setTenv(TEnv))) .
          ( #freezeKVariables(T, setTenv(TEnv)):>Type
          )
      ]
    </tenv>
```

As for *unify*, we can take advantage of the build-in pattern matching capabilities:

```
syntax KItem ::= Type "=Type" Type
rule T =Type T => .
```

By using a new function `=Type` with the rewriting rule `rule T =Type T => .` we can force the system to pattern match when ever we need to. Note that if we do not use this trick, the system will think that all existentially quantified variables are type variables and will therefore stop midway.

References

[Mil78] Robin Milner. "A theory of type polymorphism in programming". In: *Journal of Computer and System Sciences* 17 (1978), pp. 348–375.

[RS14] Grigore Rosu and Traian-Florin Serbanuta. "K Overview and SIMPLE Case Study". In: *Electr. Notes Theor. Comput. Sci.* 304 (2014), pp. 3–56. DOI: 10.1016/j.entcs.2014.05.002. URL: https://doi.org/10.1016/j.entcs.2014.05.002.