3.3 Type Inference

In the first section of this chapter we defined a type system, in the second section we introduced a syntax for our language. Now we want to define rules how to obtain the type of a given program written in our language.

3.3.1 Typing Judgments

A type system is a set of inference rules to derive various kinds of typing judgments. These *inference rules* have the following form

$$\frac{P_1 \dots P_n}{C}$$

where the judgments P_1 up to P_n are the premises of the rule and the judgment C is its conclusion.

We can read it in two ways:

- "If all premises hold then the conclusion holds as well" or
- "To prove the conclusion we need to prove all premises".

We will now provide a judgment for every production rule defined in the last section. Ultimately, we will have a judgment $p : T$ which indicates that a program p is of a type T and therefore well-formed.

If the type T is known then we talk about *type checking* else we call the process of finding the judgment *type inference*.

TYPE SIGNATURE JUDGMENTS

For type signature judgments, let Γ be a type context, $T \in \mathcal{T}$ and $a_i \in \mathcal{V}, T_i \in \mathcal{T}$ for all $i \in \mathbb{N}_1^n$ and $n \in \mathbb{N}$.

For $llv \in$ `<list-lower-var>` the judgment has the form

$$llv : (a_1, \dots, a_n)$$

which can be read as "llv defines the list $(a_1, \dots, a_n)$".

For $ltf \in$ `<list-type-fields>` the judgment has the form

$$\Gamma \vdash ltf : \{a_1 : T_1, \dots, a_n : T_n\}$$

which can be read as "given Γ, ltf has the type $\{a_1 : T_1, \dots, a_n : T_n\}$".

For $lt \in$ `<list-type>` the judgment has the form

$$\Gamma \vdash lt : (T_1, \dots, T_n)$$

which can be read as "given Γ, lt defines the list $(T_1, \dots, T_n)$".

For $t \in$ `<type>` the judgment has the form

$$\Gamma \vdash t : T$$

which can be read as "given Γ, t has the type T".

PATTERN JUDGMENTS

For pattern judgments, let Γ, Δ and Θ be type contexts. Let $T \in \mathcal{T}$ and $T_i \in \mathcal{T}, a_i \in \mathcal{V}$ for all $i \in \mathbb{N}_0^n$ and $n \in \mathbb{N}$.

For $lpl \in$ `<list-pattern-list>` the judgment has the form

$$\Gamma, \Delta \vdash: \text{match}_\Theta(\mathit{List}\ T, \mathit{lpl})$$

which can be read as "given Γ, Δ, we can match $\mathit{List}\ T$ with the pattern lpl by using the context Θ".

For $lps \in$ `<list-pattern-sort>` the judgment has the form

$$\Gamma, \Delta \vdash \text{match}_\Theta((T_1, \dots, T_n), \mathit{lps})$$

which can be read as "given Γ and Δ, we can match $(T_1, \dots, T_n)$ with the pattern lps by using the context Θ".

For $lpv \in$ `<list-pattern-vars>` the judgment has the form

$$\mathit{lpv} : (a_1, \dots, a_n)$$

which can be read as "lpv defines the list $(a_1, \dots, a_n)$".

For $p \in$ `<pattern>` the judgment has the form

$$\Gamma, \Delta \vdash \text{match}_\Theta(T, p)$$

which can be read as "given Γ and Δ, we can match T with the pattern p by using the context Θ".

EXPRESSION JUDGMENTS

For expression judgments, let Γ, Δ be type contexts, $T \in \mathcal{T}$, $a \in \mathcal{V}$ and $T_i \in \mathcal{T}, a_i \in \mathcal{V}$ for all $i \in \mathbb{N}_0^n, n \in \mathbb{N}$.

For $\mathit{lef} \in$ `<list-exp-field>` the judgment has the form

$$\Gamma, \Delta \vdash \mathit{lef} : \{a_1 : T_1, \dots, a_n : T_n\}$$

which can be read as "given Γ and Δ, lef has the type $\{a_1 : T_1, \dots, a_n : T_n\}$".

For $\mathit{mes} \in$ `<maybe-exp-sign>` the judgment has the form

$$\Gamma, \mathit{mes} \vdash a : T$$

which can be read as "given Γ, a has the type T under the assumption mes".

For $lc \in$ `<list-case>` the judgment has the form

$$\Gamma, \Delta, T_1 \vdash \mathit{lc} : T_2$$

which can be read as "given Γ and Δ and a type T_1, lc has the type T_2".

For $b \in$ `<bool>` the judgment has the form

$$b : T$$

which can be read as "b has the type T".

For $i \in$ `<int>` the judgment has the form

$$e : T$$

which can be read as "i has the type T".

For $le \in$ `<list-exp>` the judgment has the form

$$\Gamma, \Delta \vdash le : List\,T$$

which can be read as "given Γ and Δ, le has the type $List\,T$".

For $e \in$ `<exp>` the judgment has the form

$$\Gamma, \Delta \vdash e : T$$

which can be read as "given Γ and Δ, e is of type T".

STATEMENT JUDGMENTS

For statement judgments, let $\Gamma, \Gamma_1, \Gamma_2, \Delta, \Delta_1, \Delta_2$ be a type contexts, $T, T_1, T_2 \in \mathcal{T}$, $a \in \mathcal{V}$ and $T_i, A_i \in \mathcal{T}, a_i \in \mathcal{V}$ for $i \in \mathbb{N}_0^n$ and $T_{i,j} \in \mathcal{T}$ for $i \in \mathbb{N}_0^n, n \in \mathbb{N}, j \in \mathbb{N}_0^{k_i}$ and $k_i \in \mathbb{N}$.

For $lss \in$ `<list-statement-sort>` the judgment has the form

$$lss : (c_1 : (T_{1,1}, \ldots, T_{1,k_1}), \ldots, c_n : (T_{n,1}, \ldots, T_{n,k_n}))$$

which can be read as "lss is a tuple of sorts c_i for $i \in \mathbb{N}_1^n$ such that that each define a list $(T_{i,1}, \ldots, T_{i,k_i})$.

For $lsv \in$ `<list-statement-var` the judgment has the form

$$lsv : (a_1, \ldots, a_n)$$

which can be read as "lsv describes the list $(a_1, \ldots, a_n)$".

For $ls \in$ `<list-statement>` the judgment has the form

$$\Gamma_1, \Delta_2, ls \vdash \Gamma_2, \Delta_2$$

which can be read as "the list of statements ls maps Γ_1 to Γ_2 and Δ_1 to Δ_2".

For $mss \in$ `<maybe-statement-sign>` the judgment has the form

$$\Gamma, mss \vdash a : T$$

which can be read as "given Γ, a has the type T_2 under the assumption mss".

For $s \in$ `<statement>` the judgment has the form

$$\Gamma_1, \Delta_1, s \vdash \Gamma_2, \Delta_2$$

which can be read as "the statement s maps Γ_1 to Γ_2 and Δ_1 to Δ_2".

For $mms \in$ `<maybe-main-sign>` the judgment has the form

$$\Gamma, mms \vdash \text{main} : T$$

which can be read as "the main function has type T under the assumtion mms".

For $prog \in$ `<program>` the judgment has the form

$$prog : T$$

which can be read as "the program $prog$ is wellformed and has the type T".

3.3.2 Auxiliary Definitions

We will assume that T is a mono type, T is a type variable and $T_1 = T_2$ denotes the equiality of two given types T_1 and T_2.

We will write $a_1, \ldots, a_n = \text{free}(T)$ to denote all free variables $a_1, \ldots, a_n$ of T.

INSTANTIATION, GENERALIZATION

The type system that we are using is polymorphic, meaning that whenever a judgment holds for a type, it will also hold for a more specific type. To counter this polymorphism we will force the types in a judgment to be unique by explicitly stating whenever we want to use a more specific or general type.

Definition 3.1: Instantiation

Let $\Delta : \mathcal{V} \nrightarrow \mathcal{T}$ be a type context,$T \in \mathcal{T}$ and e be an expression.

Then we define

$$\Delta(e) \sqsubseteq T :\Leftrightarrow \exists T_0 \in \mathcal{T}.(e, T_0) \in \Delta \wedge T_0 \sqsubseteq T$$

Note that Δ is a partial function and therefore $\Delta(e)$ would only be defined if T_0 exists. If T_0 does no exist, then this predicate will be false.

The act of replacing T_0 with the more specific type T is called *Instantiation* and is typically in the text books introduced as an additional inference rule.

Definition 3.2: Generalization

Let Δ_1, Δ_2 be type contexts, $a \in \mathcal{V}$. Let $T, T' \in \mathcal{T}$ such that T' is a mono type $\forall c_1. \ldots . \forall c_m T' = T$ for some $c_i \in \mathcal{V}, i \in \mathbb{N}_0^m$.

We define

$$\text{insert}_{\Delta_1}(\Delta_2) :=$$
$$\Delta_1 \cup \left\{ \; (a, \forall b_1 \ldots \forall b_n . T') \;\middle|\; \begin{array}{l} (a, T) \in \Delta_2 \\ \wedge \{b_1, \ldots, b_n\} = \{b \mid b \in \text{free}(T) \wedge (b, \_) \notin \Delta_2\} \end{array} \right\}$$

This definition essentially states that all quantified variables of T, that occur in Δ_2, will be dropped and any free variables will be quantified. The act of removing a quantified variable that is already in the type context is called *Generalization* and is also typically found as an inference rule in text books.

PREDEFINED TYPES

Additionally, we define

$$\begin{aligned} Bool &:= \mu\_.\, True | False \\ Nat &:= \mu C.1 | Succ\; C \\ Int &:= \mu\_.0 \mid Pos\; Nat \mid Neg\; Nat \\ List &:= \forall a.\mu C.[\,]\,| Cons\; a\; C \end{aligned}$$

3.3.3 Inference Rules for Type Signatures

LIST-LOWER-VAR

Judgment: $llv : (a_1, \ldots, a_n)$

$$\texttt{""} : ()$$

For an empty list we return the empty tuple.

$$\frac{llv : (a_1, \ldots, a_n) \quad (a_0, a_1, \ldots, a_n) = T}{a_0\; llv : T}$$

For a nonempty list, we append the head a to the type T of the tail l.

LIST-TYPE-FIELDS

Judgment: $\Gamma \vdash ltf : \{a_1 : T_1, \ldots, a_n : T_n\}$

$$\Gamma \vdash \texttt{""} : \{\}$$

$$\frac{\Gamma \vdash t : T_0 \quad \Gamma \vdash ltf : \{a_1 : T_1, \ldots, a_n : T_n\} \quad \{a_0 : T_0, a_1 : T_1, \ldots, a_n : T_n\} = T}{\Gamma \vdash a_0\; \texttt{":"}\; t\; \texttt{","}\; ltf : T}$$

The type context Γ is used in the judgment $\Gamma \vdash t : T_0$ that turns the type signature t into a type T_0.

LIST-TYPE

Judgment: $\Gamma \vdash lt : (T_1, \ldots, T_n)$

$$\Gamma \vdash \texttt{""} : ()$$

$$\frac{\Gamma \vdash t : T_0 \quad \Gamma \vdash lt : (T_1, \ldots, T_n) \quad (T_0, T_1, \ldots, T_n) = T}{\Gamma \vdash t\, lt : T}$$

TYPE

Judgment: $\Gamma \vdash t : T$

$$\frac{\mathit{Bool} = T}{\Gamma \vdash \texttt{"Bool"} : T}$$

$$\frac{\mathit{Int} = T}{\Gamma \vdash \texttt{"Int"} : T}$$

$$\frac{\mathit{List}\ T_2 = T_1 \quad \Gamma \vdash t : T_2}{\Gamma \vdash \texttt{"List" t} : T_1}$$

$$\frac{(T_1, T_2) = T_0 \quad \Gamma \vdash t_1 : T_1 \quad \Gamma \vdash t_2 : T_2}{\Gamma \vdash \texttt{"("}\ t_1\ \texttt{","}\ t_2\ \texttt{")"} : T_0}$$

$$\frac{\Gamma \vdash ltf : T}{\Gamma \vdash \texttt{"\{"}\ ltf\ \texttt{"\}"} : T}$$

$$\frac{T_1 \to T_2 = T_0 \quad \Gamma \vdash t_1 : T_1 \quad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1 \to t_2 : T_0}$$

$$\frac{(c, T') \in \Gamma \quad \Gamma \vdash l : (T_1, \ldots, T_n) \quad \overline{T'}\ T_1 \ldots T_n = T}{\Gamma \vdash c\ l : T}$$

For a given type T we write the application constructor as $\overline{T}$.

$$\frac{\forall a.a = T}{\Gamma \vdash a : T}$$

Example 3.1

In example **??** we have looked at the syntax for a list reversing function.

The type signiture for the `reverse` function was `List a -> List a`. We will now show how we can obtain the curresponding type T_0. For that, let $\Gamma = \varnothing$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\forall a.a = T_3}}{\varnothing \vdash a : T_3} \quad \dfrac{}{List\ T_3 = T_1}}{\varnothing \vdash Lista : T_1} \quad \dfrac{\dfrac{\dfrac{}{\forall a.a = T_4}}{\varnothing \vdash a : T_4} \quad \dfrac{}{List\ T_4 = T_2}}{\varnothing \vdash Lista : T_2}}{T_1 \to T_2 = T_0}}{\varnothing \vdash \texttt{List a -> List a} : T_0}$$

We can therefore conclude that $T_0 = List\ (\forall a.a) \to List\ (\forall a.a) = \forall a.List\ a \to List\ a$.

3.3.4 Inference Rules for patterns

LIST-PATTERN-LIST

Judgment: $\Gamma, \Delta \vdash: \text{match}_\Theta(List\ T, lpl)$

$$\Gamma, \Delta \vdash: \text{match}_\varnothing(\forall a.List\ a, \texttt{""})$$

$$\dfrac{\begin{array}{c}\Gamma, \Delta \vdash: \text{match}_{\Theta_1}(T, p) \\ \Gamma, \Delta \vdash: \text{match}_{\Theta_2}(List\ T, lpl) \quad \Theta_1 \cap \Theta_2 = \varnothing \quad \Theta_3 = \text{insert}_{\Theta_1}(\Theta_2)\end{array}}{\Gamma, \Delta \vdash: \text{match}_{\Theta_3}(List\ T, p\ \texttt{","}\ lpl)}$$

Θ_3 is the set of all bindings in the list with head p and tail lpl. Variables may only bound once, therefore we need to ensure that the binding Θ_1 of p and the binding Θ_2 of lpl are disjoint.

LIST-PATTERN-SORT

Judgment: $\Gamma, \Delta \vdash \text{match}_\Theta((T_1, \dots, T_n), lps)$

$$\Gamma, \Delta \vdash: \text{match}_\Theta((), \texttt{""})$$

$$\dfrac{\begin{array}{c}\Gamma, \Delta \vdash: \text{match}_{\Theta_1}(T_0, p) \\ \Gamma, \Delta \vdash: \text{match}_{\Theta_2}((T_1, \dots, T_n), lps) \quad \Theta_1 \cap \Theta_2 = \varnothing \quad \Theta_3 = \text{insert}_{\Theta_1}(\Theta_2)\end{array}}{\Gamma, \Delta \vdash: \text{match}_{\Theta_3}((T_0, T_1, \dots, T_n), p\ lps)}$$

LIST-PATTERN-VARS

Judgment: $lpv : (a_1, \dots, a_n)$

$$\texttt{""} : ()$$

$$\frac{lpv : (a_1, \ldots, a_n)}{a_0 \texttt{ "" } lpv : (a_0, a_1, \ldots, a_n)}$$

PATTERN

$$\frac{b : \mathit{Bool}}{\Gamma, \Delta \vdash \text{match}_{\varnothing}(\mathit{Bool}, b)}$$

$$\frac{i : \mathit{Int}}{\Gamma, \Delta \vdash \text{match}_{\varnothing}(\mathit{Int}, i)}$$

$$\frac{\Gamma, \Delta \vdash \text{match}_{\Theta}(\mathit{List}\ T, lpl)}{\Gamma, \Delta \vdash \text{match}_{\Theta}(\mathit{List}\ T, \texttt{"["}lpl\texttt{"]"})}$$

$$\frac{\Gamma, \Delta \vdash \text{match}_{\Theta_1}(T_1, p_1) \quad \Gamma, \Delta \vdash \text{match}_{\Theta_2}(T_2, p_2) \\ \Theta_1 \cap \Theta_2 = \varnothing \quad \text{insert}_{\Theta_1}(\Theta_2) = \Theta_3}{\Gamma, \Delta \vdash \text{match}_{\Theta_3}((T_1, T_2), \texttt{"("}p_1\texttt{","}p_2\texttt{")"})}$$

$$\frac{\Delta(c) \sqsubseteq T_1 \to \cdots \to T_n \to T_0 \quad \Gamma, \Delta \vdash \text{match}_{\Theta}((T_1, \ldots, T_n), lps)}{\Gamma, \Delta \vdash \text{match}_{\Theta}(T_0, c\ lps)}$$

$$\frac{(a, \_) \notin \Delta \quad \Theta = \{(a, T)\}}{\Gamma, \Delta \vdash \text{match}_{\Theta}(T, a)}$$

$$\frac{(a, \_) \notin \Delta \quad (a, \_) \notin \Theta_1 \quad \text{insert}_{\Theta_1}(\{(a, T)\}) = \Theta_2 \quad \Gamma, \Delta \vdash \text{match}_{\Theta_1}(T, p)}{\Gamma, \Delta \vdash \text{match}_{\Theta_2}(T, p\texttt{"as"}a)}$$

$$\frac{lpv = (a_1, \ldots, a_n) \quad T = \{a_1 : T_1, \ldots, a_n : T_n\} \\ \Delta \cap \{(a_1, \_), \ldots, (a_n, \_)\} = \varnothing \quad \Theta = \{(a_1, T_1), \ldots, (a_n, T_n)\}}{\Gamma, \Delta \vdash \text{match}_{\Theta}(T, \texttt{"\{"}lpv\texttt{"\}"})}$$

$$\frac{\Gamma, \Delta \vdash \text{match}_{\Theta_1}(T, p_1) \quad \Gamma, \Delta \vdash \text{match}_{\Theta_2}(\mathit{List}\ T, p_2) \\ \Theta_1 \cap \Theta_2 = \varnothing \quad \text{insert}_{\Theta_1}(\Theta_2) = \Theta_3}{\Gamma, \Delta \vdash \text{match}_{\Theta_3}(\mathit{List}\ T, p_1 \texttt{ "::" } p_2)}$$

$$\Gamma, \Delta \vdash \text{match}_{\varnothing}(\forall a.a, \texttt{"\_"})$$

Example 3.2

In example **??** we have looked at the syntax for list reversing function. We will now find the bindings Θ_0 for the following pattern used in the reversing function.

```
a :: _
```

We assume that the type of the expression being matched is $List\ Int$ and $\Gamma = \Delta = \varnothing$.

$$\dfrac{\dfrac{\dfrac{}{(a,\_) \notin \varnothing} \quad \dfrac{}{\Theta_1 = \{(a, \mathit{Int})\}}}{\Gamma, \Delta \vdash \text{match}_{\Theta_1}(\mathit{Int}, a)} \quad \dfrac{\dfrac{\overline{\Theta_2 = \varnothing}}{\Gamma, \Delta \vdash \text{match}_{\Theta_2}(\forall a.a, \_)}}{\Gamma, \Delta \vdash \text{match}_{\Theta_2}(\mathit{List\ Int}, \_)} \quad \dfrac{}{\text{insert}_{\Theta_1}(\Theta_2) = \Theta_0} \quad \Theta_1 \cap \Theta_2 = \varnothing}{\varnothing, \varnothing \vdash \text{match}_{\Theta_0}(\mathit{List\ Int}, \texttt{a " :: \_"})}$$

After ensuring $\Theta_1 \cap \Theta_2 = \{(a, \mathit{Int})\} \cap \varnothing = \varnothing$ we can conclude

$$\Theta_0 = \text{insert}_{\{(a,\mathit{Int})\}}(\varnothing) = \{(a, \mathit{Int})\}.$$

3.3.5 Inference Rules for Expressions

LIST-EXP-FIELD

Judgment: $\Gamma, \Delta \vdash lef : \{a_1 : T_1, \ldots, a_n : T_n\}$

$$\frac{\Gamma, \Delta \vdash e : T}{\Gamma, \Delta \vdash a \texttt{ "=" } e : \{a : T\}}$$

$$\frac{\Gamma, \Delta \vdash lef : T \quad \Gamma, \Delta \vdash e : T_0 \quad \{a_0 : T_0, \ldots, a_n : T_n\} = T}{\Gamma, \Delta \vdash a_0 \texttt{ "=" } e \texttt{ "," } lef : T}$$

MAYBE-EXP-SIGN

Judgment: $\Gamma, mes \vdash a : T$

$$\Gamma, \texttt{""} \vdash a : T$$

If no argument is given, then we do nothing.

$$\frac{\Gamma \vdash t : T a_1 = a_2}{\Gamma, a_1 \texttt{":"} \texttt{"}t\texttt{";"} \vdash a_2 : T}$$

If we have a variable a_1 and a type T, then the variables a_2 need to match. The type signature t defines the type of a_2.

LIST-CASE

Judgment: $\Gamma, \Delta, T_1 \vdash lc : T_2$

$$\frac{\Gamma, \Delta \vdash \text{match}_{\Theta}(T_1, p) \quad \Gamma, \text{insert}_{\Delta}(\Theta) \vdash e : T_2}{\Gamma, \Delta, T_1 \vdash p \texttt{"->"}\ e : T_2}$$

Given the type T_1 of the expression that is being matched, we can now find all new binding Θ by matching p with T_1. Finally, we unify Δ with Θ.

$$\frac{\Gamma, \Delta \vdash \text{match}_{\Theta}(T_1, p) \quad \Gamma, \text{insert}_{\Delta}(\Theta) \vdash e : T_2 \quad \Gamma, \Delta, T_1 \vdash lc : T_2}{\Gamma, \Delta, T_1 \vdash p \texttt{"->"}\ e \texttt{";"}\ lc : T_2}$$

BOOL

Judgment: $b : T$

$$b : \mathit{Bool}$$

INT

Judgment: $i : T$

$$i : \mathit{Int}$$

We have proven in theorem **??** that Nat is isomorph to $\mathbb{N}$. Is should be trivial to therefore conclude that Int is isomorph to $\mathbb{Z}$. And therefore this rule is justified.

LIST-EXP

Judgment: $\Gamma, \Delta \vdash le : \mathit{List}\ T$

$$\Gamma, \Delta \vdash \texttt{""} : \forall a.\mathit{List}\ a$$

$$\frac{\Gamma, \Delta \vdash e : T \quad \Gamma, \Delta \vdash le : \mathit{List}\ T}{\Gamma, \Delta \vdash e \texttt{","}\ le : \mathit{List}\ T}$$

EXP

Judgment: $\Gamma, \Delta \vdash e : T$

$$\Gamma, \Delta \vdash \texttt{"foldl"} : \forall a.\forall b.(a \to b \to b) \to b \to \mathit{List}\ a \to b$$

$$\Gamma, \Delta \vdash \texttt{"(::)"} : \forall a.a \to \mathit{List}\ a \to \mathit{List}\ a$$

$$\Gamma, \Delta \vdash \texttt{"(+)"} : \mathit{Int} \to \mathit{Int} \to \mathit{int}$$

$$\Gamma, \Delta \vdash \texttt{"(-)"} : \mathit{Int} \to \mathit{Int} \to \mathit{int}$$

$$\Gamma, \Delta \vdash \texttt{"(*)"} : \mathit{Int} \to \mathit{Int} \to \mathit{int}$$

$$\Gamma, \Delta \vdash \texttt{"(//)"} : \mathit{Int} \to \mathit{Int} \to \mathit{int}$$

$$\Gamma, \Delta \vdash \texttt{"(<)"} : \mathit{Int} \to \mathit{Int} \to \mathit{Bool}$$

$$\Gamma, \Delta \vdash \texttt{"(==)"} : \mathit{Int} \to \mathit{Int} \to \mathit{Bool}$$

$$\Gamma, \Delta \vdash \texttt{"not"} : \mathit{Bool} \to \mathit{Bool}$$

$$\Gamma, \Delta \vdash \texttt{"(\&\&)"} : \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool}$$

$$\Gamma, \Delta \vdash \texttt{"(||)"} : \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool}$$

$$\frac{\Gamma, \Delta \vdash e_1 : T_1 \quad \Gamma, \Delta \vdash e_2 : T_1 \to T_2}{\Gamma, \Delta \vdash e_1 \; \texttt{"|>"} \; e_2 : T_2}$$

$$\frac{\Gamma, \Delta \vdash e_1 : T_1 \to T_2 \quad \Gamma, \Delta \vdash e_2 : T_2 \to T_3}{\Gamma, \Delta \vdash e_1 \; \texttt{">>"} \; e_2 : T_1 \to T_3}$$

$$\frac{\Gamma, \Delta \vdash e_1 : \mathit{Bool} \quad \Gamma, \Delta \vdash e_2 : T \quad \Gamma, \Delta \vdash e_3 : T}{\Gamma, \Delta \vdash \texttt{"if"} \; e_1 \; \texttt{"then"} \; e_2 \; \texttt{"else"} \; e_3 : T}$$

$$\frac{\Gamma, \Delta \vdash \mathit{lef} : \{a_1 : T_1, \ldots, a_n : T_n\}}{\Gamma, \Delta \vdash \texttt{"\{"}\mathit{lef}\texttt{"\}"} : \{a_1 : T_1, \ldots, a_n : T_n\}}$$

$$\Gamma, \Delta \vdash \texttt{"\{\}"} : \{\}$$

$$\frac{\begin{array}{c}\Gamma, \Delta \vdash \mathit{lef} : \{a_1 : T_1, \ldots, a_n : T_n\} \\ \Gamma, \Delta \vdash \Delta(a) \sqsubseteq T_0 \quad T_0 = \{a_1 : T_1, \ldots, a_n : T_n, \ldots\}\end{array}}{\Gamma, \Delta \vdash \texttt{"\{"} \; a \; \texttt{"|"} \; \mathit{lef} \; \texttt{"\}"} : T_0}$$

Since Elm version 0.19, released in 2018, setters are not allowed to change the type of a field in a record.

$$\frac{(a_1, \{a_2 : T, \dots\}) \in \Delta}{\Gamma, \Delta \vdash a_1 \texttt{"."} a_2 : T}$$

$$\frac{(a, \_) \notin \Delta \quad \Gamma, \Delta \vdash e_1 : T_1 \quad mes : T_1 \vdash a : T_1 \\ \Gamma, \text{insert}_\Delta(\{(a, T_1)\}) \vdash e_2 : T_2}{\Gamma, \Delta \vdash \texttt{"let"}\ mes\ a \texttt{"="}\ e_1\ \texttt{"in"}\ e_2 : T_2}$$

$$\frac{\Gamma, \Delta \vdash e_1 : T_1 \quad \Gamma, \Delta, T_1 \vdash lc : T_2}{\Gamma, \Delta \vdash \texttt{"case"}\ e_1\ \texttt{"of"}\ \ \texttt{"["}\ lc\ \texttt{"]"} : T_2}$$

$$\frac{\Gamma, \Delta \vdash e_1 : T_1 \to T_2 \quad \Gamma, \Delta \vdash e_2 : T_1}{\Gamma, \Delta \vdash e_1\ e_2 : T_2}$$

$$\frac{b : T}{\Gamma, \Delta \vdash b : T}$$

$$\frac{i : T}{\Gamma, \Delta \vdash i : T}$$

$$\frac{\Gamma, \Delta \vdash le : T}{\Gamma, \Delta \vdash \texttt{"["} le \texttt{"]"} : T}$$

$$\frac{\Gamma, \Delta \vdash e_1 : T_1 \quad \Gamma, \Delta \vdash e_2 : T_2}{\Gamma, \Delta \vdash \texttt{"("}\ e_1\ \texttt{","}\ e_2 \texttt{")"} : (T_1, T_2)}$$

$$\frac{\Gamma, \Delta \vdash \text{match}_\Theta(T_1, p) \quad \Gamma, \text{insert}_\Delta(\Theta) \vdash e : T_2}{\Gamma, \Delta \vdash \texttt{"\textbackslash"} p \texttt{"->"} e : T_1 \to T_2}$$

In Elm function arguments may be pattern matched, this mostly used to “unwrap” a type, meaning to bind contained elements to variables.

$$\frac{\Delta(c) \sqsubseteq T}{\Gamma, \Delta \vdash c : T}$$

$$\frac{\Delta(a) \sqsubseteq T}{\Gamma, \Delta \vdash a : T}$$

Example 3.3

In example **??** we have looked at the syntax for a list reversing function. We can now check the type $T_0 = \forall a. List\ a \to List\ a$ of the `reverse` function for $\Gamma = \Delta = \varnothing$, $\Delta = \varnothing$. The body of the *reverse* function is as follows:

```
foldl (::) []
```

$$\dfrac{\dfrac{\dfrac{}{\varnothing,\varnothing \vdash \texttt{"foldl"} : T_2} \quad \dfrac{}{\varnothing,\varnothing \vdash \texttt{"(::)"} : \forall a.\mathit{List}\ a \to \mathit{List}\ a}}{\varnothing,\varnothing \vdash \texttt{"foldl (::)"} : T_1} \quad \dfrac{\dfrac{}{\varnothing,\varnothing \vdash \texttt{""} : \forall a.a}}{\varnothing,\varnothing \vdash \texttt{"[]"} : \forall a.\mathit{List}\ a}}{\varnothing,\varnothing \vdash \texttt{"foldl (::) []"} : T_0}$$

where $T_1 = \forall a.\mathit{List}\ a \to \mathit{List}\ a \to \mathit{List}\ a$ and $T_2 = \forall a.(\mathit{List}\ a \to \mathit{List}\ a) \to \mathit{List}\ a \to \mathit{List}\ a \to \mathit{List}\ a$.

3.3.6 Inference Rules for Statements

LIST-STATEMENT-VAR

Judgment: $\mathit{lsv} : (a_1, \ldots, a_n)$

$$\texttt{""} : ()$$

$$\frac{\mathit{lsv} : (a_1, \ldots, a_n)}{a_0\ \mathit{lsv} : (a_0, a_1, \ldots, a_n)}$$

LIST-STATEMENT-SORT

Judgment: $\mathit{lss} : (c_1 : (T_{1,1}, \ldots, T_{1,k_1}), \ldots, c_n : (T_{n,1}, \ldots, T_{n,k_n}))$

$$\frac{\Gamma \vdash \mathit{lt} : (T_0, \ldots, T_n)}{c\ \mathit{lt} : (c : (T_0, \ldots, T_n))}$$

$$\frac{\Gamma \vdash \mathit{lt} : (T_{0,1}, \ldots, T_{0,k_n}) \quad \mathit{lss} : \begin{pmatrix} a_1 : (T_{1,1}, \ldots, T_{1,k_1}), \\ \vdots \\ a_n : (T_{n,1}, \ldots, T_{n,k_n}) \end{pmatrix}}{c\ \mathit{lt}\ \texttt{"|"}\ \mathit{lss} : \begin{pmatrix} a_0 : (T_{0,1}, \ldots, T_{0,k_0}), \\ a_1 : (T_{1,1}, \ldots, T_{1,k_1}), \\ \vdots \\ a_n : (T_{n,1}, \ldots, T_{n,k_n}) \end{pmatrix}}$$

LIST-STATEMENT

Judgment: $\Gamma_1, \Delta_1, \mathit{ls} \vdash \Gamma_2, \Delta_2$

$$\frac{\Gamma_1 = \Gamma_2 \quad \Delta_1 = \Delta_2}{\Gamma_1, \Delta_1 \texttt{""} \vdash \Gamma_2, \Delta_2}$$

$$\frac{\Gamma_1, \Delta_1, s \vdash \Gamma_2, \Delta_2 \quad \Gamma_2, \Delta_2, \mathit{ls} \vdash \Gamma_3, \Delta_3}{\Gamma_1, \Delta_1, s\ \texttt{";"}\ \mathit{ls} \vdash \Gamma_3, \Delta_3}$$

MAYBE-STATEMENT-SIGN

Judgment: $\Gamma, mss \vdash a : T$

$$\Gamma, \texttt{""} \vdash a : T$$

$$\frac{\Gamma \vdash t : T a_1 = a_2}{\Gamma, a_1 \texttt{ ":" } t \texttt{ ";" } \vdash a_2 : T}$$

STATEMENT

Judgment: $\Gamma_1, \Delta_1, s \vdash \Gamma_2, \Delta_2$

$$\frac{\begin{array}{c}\Gamma_1 = \Gamma_2 \quad (a, \_) \notin \Delta_1 \\ \Gamma_1, mss \vdash e : T \quad \Gamma_1, \Delta_1 \vdash e : T \quad \Delta_2 = \text{insert}_{\Delta_1}(\{(a, T)\})\end{array}}{\Gamma_1, \Delta_1, mss\ a \texttt{"="} e \vdash \Gamma_2, \Delta_2}$$

$$\frac{\begin{array}{c}\Delta_1 = \Delta_2 \quad (c, \_) \notin \Gamma_1 \quad \Gamma \vdash t : T_1 \\ T_2 \text{ is a mono type} \quad lsv : (a_1, \ldots, a_n) \quad \{a_1 \ldots a_n\} = \text{free}(T_2) \\ \forall a_1. \ldots \forall a_n. T_2 = T_1 \quad \Gamma_2 = \Gamma_2 \cup \{(c, T_1)\}\end{array}}{\Gamma_1, \Delta_1, \texttt{"type alias"}\ c\ lsv \texttt{ "=" } t \vdash \Gamma_2, \Delta_2}$$

$$\frac{\begin{array}{c}(c, \_) \notin \Gamma_1 \quad lsv : (a_1, \ldots, a_n) \\ lss : (c_1 : (T_{1,1}, \ldots, T_{1,k_1}), \ldots, c_n : (T_{n,1}, \ldots, T_{n,k_n})) \\ \Delta_1 \cap \{(c_1, \_), \ldots, (c_n, \_)\} = \varnothing \quad \{a_1 \ldots a_n\} = \text{free}(T_2) \\ \mu C. c_1\ T_{1,1}\ \ldots\ T_{1,k_1} \mid \ldots \mid c_n\ T_{n,1}\ \ldots\ T_{n,k_n} = T_2 \quad \forall a_1. \ldots \forall a_n. T_2 = T_1 \\ \Gamma_1 \cup \{(c, T_1)\} = \Gamma_2 \quad \text{insert}_{\Delta_1}(\left\{\begin{array}{l}(c_1, T_{1,1} \to \cdots \to T_{1,k_1} \to T_1), \\ \vdots \\ (c_n, T_{n,1} \to \cdots \to T_{n,k_n} \to T_1)\end{array}\right\}) = \Delta_2\end{array}}{\Gamma_1, \Delta_1, \texttt{"type"}\ c\ lsv \texttt{"="} lss \vdash \Gamma_2, \Delta_2}$$

The list *lss* provides us with the structure of the type. From there we construct the type T_2 and bind all variables, thus creating the poly type T_1. Additionally, every sort c_i for $i \in \mathbb{N}_1^n$ has its own constructor that gets added to Δ_1 under the name c_i. In Elm these constructors are the only constants beginning with an upper-case letter.

MAYBE-MAIN-SIGN

Judgment: $\Gamma, mms \vdash \text{main} : T$

$$\Gamma, \texttt{""} \vdash \text{main} : T$$

$$\frac{\Gamma \vdash t : T}{\Gamma, \texttt{"main :"} t \texttt{";"} \vdash \text{main} : T}$$

PROGRAM

Judgment: *prog* : T

$$\frac{\varnothing, \varnothing ls \vdash \Gamma, \Delta \quad \Gamma, mms \vdash \mathsf{main} : T \quad \Gamma, \Delta \vdash e : T}{ls\ mms\ \texttt{"main = "}\ e : T}$$