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

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

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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 $lef \in$ `<list-exp-field>` the judgment has the form

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

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

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

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

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

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

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".

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

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

$$\textit{prog} : T$$

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

3.3.2 Auxiliary Definitions

We will use $(e, T) \in \Gamma$ and $(e, T) \in \Delta$ to denote that a tuple (e, T) exists in Γ or Δ. We will also sometimes use a wildcard _ instead of a T if we are only interested in e.

We will use $T_1 \sqsubseteq T_2$ for given types T_1, T_2 to denote that T_1 is more general than T_2.

We will use T is a mono type, T is a type variable and type equivalence $T_1 = T_2$ for two given types T_1 and T_2.

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

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: $\Gamma \vdash t : T$

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

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

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

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

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

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

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

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 $\Gamma = \varnothing$.

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

We can therefore conclude that $T_0 = \mathit{List}\ (\forall a.a) \rightarrow \mathit{List}\ (\forall a.a) = \forall a.\mathit{List}\ a \rightarrow \mathit{List}\ a$.

3.3.4 Inference Rules for patterns

LIST-PATTERN-LIST

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

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

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

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

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

$$\frac{\Gamma, \Delta \vdash: \text{match}_{\Theta_1}(T_0, p) \qquad \Gamma, \Delta \vdash: \text{match}_{\Theta_2}((T_1, \ldots, T_n), lps) \quad \Theta_1 \cap \Theta_2 = \varnothing \quad \Theta_3 = \Theta_1 \cup \Theta_2}{\Gamma, \Delta \vdash: \text{match}_{\Theta_3}((T_0, T_1, \ldots, T_n), p\ lps)}$$

LIST-PATTERN-VARS

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

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

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

PATTERN

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

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

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

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

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

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

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

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

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

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

—

These rules do not work on their own, as the type might be too specific or a poly type needs to be instantiated. Therefore, the Hindley-Milner type system comes with two additional rules:

$$\frac{\Gamma, \Delta \vdash match_{\Theta}(T_2, p) \quad T_2 \sqsubseteq T_1}{\Gamma, \Delta \vdash match_{\Theta}(T_1, p)} \qquad \text{[PInstantiation]}$$

Read top to bottom, this rule states that if a pattern p matches the type T_2 then we may also conclude that p matches a more specific type T_1.

$$\frac{(a, \_) \notin \Delta \quad \Gamma, \Delta \vdash match_{\Theta}(T, p)}{\Gamma, \Delta \vdash match_{\Theta}(\forall a.T, p)} \qquad \text{[PGeneralization]}$$

Read bottom up, for a poly type $\forall a.T$ we may remove the for-all-quantor if the variable a has already a binding in Δ.

Taking these two new rules into account, the resulting inference rules do not have a unique rule for every pattern any longer. This means we are able to type check but not to infer a type.

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 $\mathit{List\ Int}$ and $\Gamma = \Delta = \varnothing$.

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

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

$$\Theta_0 = \{(a, \mathit{Int})\} \cup \varnothing = \{(a, \mathit{Int})\}.$$

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.

$$\frac{\Gamma \vdash t : Ta_1 = a_2}{\Gamma, a_1 \texttt{":"} t \texttt{";"} \vdash a_2 : T}$$

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

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

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.

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

BOOL

Judgment: $b : T$

$$b : Bool$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\begin{array}{c}(a, \_) \notin \Delta \quad \Gamma, \Delta \vdash e_1 : T_1 \quad \mathit{mes} : T_1 \vdash a : T_1 \\ \Gamma, \Delta \cup (a, T_1) \vdash e_2 : T_2\end{array}}{\Gamma, \Delta \vdash \texttt{"let"} \; \mathit{mes} \; a\texttt{"="} \; e_1 \; \texttt{"in"} \; e_2 : T_2}$$

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

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

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

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

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

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

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

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

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

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

—

Same as for patterns, we also include an instantiation rule as well as a generalization rule:

$$\frac{\Gamma, \Delta \vdash e : T_2 \quad T_2 \sqsubseteq T_1}{\Gamma, \Delta \vdash e : T_1} \qquad \text{[TInstantiation]}$$

$$\frac{(a, \_) \notin \Delta \quad \Gamma, \Delta \vdash e : T}{\Gamma, \Delta \vdash e : \forall a.T} \qquad \text{[TGeneralization]}$$

Thus, our inference rule has no longer a unique rule for every expression.

We will fix this problem at the end of the chapter by introducing two different methods how to embed these two rules among the others.

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 (::) []
```

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

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

3.3.6 Inference Rules for Statements

LIST-STATEMENT-VAR

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

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

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

LIST-STATEMENT-SORT

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

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

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

LIST-STATEMENT

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

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

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

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

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

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

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

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

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