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

To state that a program e is of type T (and therefor well-formed) we write

$$e : T$$

We call such a statement a judgment. Judgments are generally written in meta-language, thus we also need to provide a system of inference rules how to infer such a judgment. These *inference rules* have the following form

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

where P_1 up to P_n are premises and C is a 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"

The premises as well as the conclusion are meta-language statements in the following form

$$A_1, \dots, A_n \vdash B.$$

Such a sequent would be generally read as "If we know $A_1, \dots, A_n$ then we can prove B".

We will now provide a judgment for every production rule defined in the last section.

TYPE SIGNITURE JUDGMENTS

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

For $l \in$ `<list-lower-var>` the judgment is

$$l : (a_1, \dots, a_n).$$

For $l \in$ `<list-type-fields>` the judgment is

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

For $l \in$ `<list-type>` the judgment is

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

For $t \in$ `<txpe>` the judgment is

$$\Gamma \vdash t : T$$

PATTERN JUDGMENTS

We will now introduce another context, this time for variables instead of types:

Definition 3.1: Variable Context

$\Delta \in \mathcal{V} \nrightarrow \mathcal{T}$ is called the *variable context*.

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

For $l \in$ `<list-pattern-list>` the judgment is

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

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

For $l \in$ `<list-pattern-sort>` the judgment is

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

For $l \in$ `<list-pattern-vars>` the judgment is

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

For $p \in$ `<pattern>` the judgment is

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

EXPRESSION JUDGMENTS

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

For $l \in$ `<list-exp-field>` the judgment is

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

For $m \in$ `<maybe-exp-sign>` the judgment is

$$m : T_1 \vdash a : T_2$$

It can be read as "given that m has the type T_1, a has the type T_2".

For $l \in$ `<list-case>` the judgment is

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

For $e \in$ `<bool>` the judgment is

$$e : T$$

For $e \in$ `<int>` the judgment is

$$e : T$$

For $l \in$ `<list-exp>` the judgment is

$$\Gamma, \Delta \vdash l : (T_1, \ldots, T_n)$$

For $e \in$ `<exp>` the judgment is

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

If the type T is known then we talk about *type checking* else we call the process of finding the judgment *type inferring*. For inferring a type (for the judgment $\Gamma, \Delta \vdash e : T$), the result is not necessary unique. Therefore, we want to find the most general type, meaning a type T_1 such that

- T_1 is an inferred type: for all $T_2 \in \mathcal{T} \wedge T_1 \sqsubseteq T_2$ the judgment $\Gamma, \Delta \vdash e : T_2$ holds.
- T_1 is sharp: for all $T_2 \in \mathcal{T} \wedge T_2 \sqsubseteq T_1$ we can find $T_3 \in \mathcal{T} \wedge T_2 \sqsubseteq T_3$ such that the judgment $\Gamma, \Delta \vdash e : T_3$ does not hold.

STATEMENT JUDGMENTS

For statement judgments, let $\Gamma, \Gamma_1, \Gamma_2$ be type contexts, $\Delta, \Delta_1, \Delta_2$ be a variable contexts, $T, T_1, T_2 \in \mathcal{T}$, $a \in \mathcal{V}$ and $T_i \in \mathcal{T}, a_i \in \mathcal{V}$ 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 $l \in$ `<list-sort>` the judgment is

$$l : (a_1 : (T_{1,1}, \ldots, T_{1,k_1}), \ldots, a_n : (T_{n,1}, \ldots, T_{n,k_1}))$$

For $l \in$ `<maybe-statement-sign>` the judgment is

$$m : T_1 \vdash a : T_2$$

For $l \in$ `<list-statement>` the judgment is

$$\Gamma_1, \Delta_2, l \vdash \Gamma_2, \Delta_2$$

For $e \in$ `<statement>` the judgment is

$$\Gamma_1, \Delta_2, e \vdash \Gamma_2, \Delta_2$$

It can be read as "the statement e maps Γ_1 to Γ_2 and Δ_1 to Δ_2".

For $m \in$ `<maybe-main-sign>` the judgment is

$$m : T_1 \vdash \text{main} : T_2$$

For $e \in$ `<program>` the judgment is

$$e : T$$

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.

3.3.3 Inference Rules for type signitures

LIST-LOWER-VAR

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

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

For an empty list we return the empty tuple.

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

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

LIST-TYPE-FIELDS

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

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

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

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

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

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

TYPE

Judgment: $\Gamma \vdash t : T$

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

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

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash \texttt{"List" t} : (\forall a.\mu C.[\,]\,|\mathit{Cons}\ a\ C)\ T}$$

The resulting type is a type application $\mathit{List}\ T$ for the type $\mathit{List} = \forall a.\mu C.[\,]\,|\mathit{Cons}\ a\ C$.

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

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

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

$$\frac{(c, f) \in \Gamma \quad \Gamma \vdash l : (T_1, \ldots, T_n) \quad f\ T_1 \ldots T_n = T}{\Gamma \vdash c\, l : T}$$

Note that Γ maps variables to application constructors.

$$\frac{(a, (T)) \in \Gamma \quad T \text{ is a type variable}}{\Gamma \vdash a : T}$$

For a given type T we write the application constructor as (T).