3.3 Type Inference

We will now discuss the methods used to infer a type from a given expression.

Definition 3.1: Type of an expression

Let $T \in \mathcal{T}$. Let Γ be a type context. Let e be arbitary.

—

We say e *is of type* T *in the context of* Γ (Notation: $e : T$):$\Leftrightarrow$

$$e \in values_\Gamma(T)$$

Next we will need the semantics of `<type>`, namely a function that maps value(`<type>`) to $\mathcal{T}$. Note that the general discussion about semantics will occur in a later chapter.

Definition 3.2: Semantics of <type>

Let $n \in \mathbb{N}$. Let t, t_1, t_2 : `<type>` and c : `<upper-var>`. Let t_i : `<type>` for all $i \in \mathbb{N}_3^n$ and v_i : `<lower-var>` for all $i \in \mathbb{N}_1^n$.Let Γ be a type context. Let $Nat = \mu C.1 \mid Succ\ C$.

—

We define

$$\begin{aligned}
[\![.]\!]_\Gamma :&\text{value}(\texttt{<type>}) \to \mathcal{T}\\
[\![\texttt{Bool}]\!]_\Gamma =&\mu_.\ True \mid False\\
[\![\texttt{Int}]\!]_\Gamma =&\mu_.0 \mid Pos\ Nat | Neg\ Nat\\
[\![\texttt{List}]\!]_\Gamma =&\forall a.\mu C.[\,] \mid Cons\ a\ C\\
[\![\texttt{"("}\ t_1\ \texttt{,}\ t_2\ \texttt{")"}]\!]_\Gamma =&\{1 : [\![t_1]\!], 2 : [\![t_2]\!]\}\\
[\![\texttt{"\{\}"}]\!]_\Gamma =&\{\}\\
[\![\texttt{"\{"}\ v_1\ \texttt{":"}\ t_1\ \texttt{","}\ \dots\ \texttt{","}\ v_n\ \texttt{":"}\ t_n\ \texttt{"\}"}]\!]_\Gamma =&\{v_1 : [\![t_1]\!]_\Gamma, \dots, v_n : [\![t_n]\!]_\Gamma\}\\
[\![t_1\ \texttt{"->"}\ t_2]\!]_\Gamma =&[\![t_1]\!]_\Gamma \to [\![t_2]\!]_\Gamma\\
[\![c\ t_1 \dots t_n]\!] =&[[\![c]\!]]_\Gamma(a_1, \dots, a_n)
\end{aligned}$$

We say $[\![.]\!]_\Gamma$ is the *interpretation* of `<type>`. For any $t \in$ `<type>`,we call $[\![t]\!]_\Gamma$ the *semantic* of t.

For this section we will also use an interpretation function for expressions

$$[\![.]\!] : \text{values}(\texttt{<statement>}) \cup \text{values}(\texttt{<expression>}) \to A$$

for some nonempty set A. We will discuss the definition of such a function as well as the definition of A in the next chapter. For now, they will be arbitrary.

Definition 3.3: Variable Context

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

Types are typically inferred using so called *typing rules* [Pie+02]. Using these rules we can construct a truth tree, who's leafs are axioms, to prove that $\Delta \vdash [\![\mathsf{e}]\!] : T$ for an expression e, a type T, a variable context Δ and a interpretation function for expressions Δ. Such a proving process is called *type checking*. It is also possible to find the most general T such that $\Delta \vdash [\![\mathsf{e}]\!] : T$ using the typing rules. This is called *type inference*.

There are two universal typing rules for any Hindley-Milner type system.

Definition 3.4: Instantiation, Generalization

Let $T', T \in \mathcal{T}$ and $e \in$ values(`<statement>`) $\cup$ values(`<expression>`). Let a be a type variable. Let Δ be a variable context. Let A be a set and $[\![.]\!]$: values(`<statement>`) $\cup$ values(`<expression>`) $\to A$

—

$$\frac{T' \sqsubseteq T \quad \Gamma, \Delta \vdash [\![\mathsf{e}]\!] : T'}{\Gamma, \Delta \vdash [\![\mathsf{e}]\!] : T} \qquad [\mathit{Instantiation}]$$

$$\frac{(a, \_) \notin \Delta \quad \Gamma, \Delta \vdash [\![\mathsf{e}]\!] : T}{\Gamma, \Delta \vdash [\![\mathsf{e}]\!] : \forall a. T} \qquad [\mathit{Generalization}]$$

The $[\mathit{Instantiation}]$ rule says that if a type can be infered, the same holds for a more specific type. The $[\mathit{Generalization}]$ rule states the opposite: if a type with a free variable can be inferred, then the same holds for a poly type, binding the free variable.

3.3.1 Typing rules for statements

The typing rules for statements are as follows.

Definition 3.5: typing rules for statements

Let $n, m \in \mathbb{N}$. Let $k_i \in \mathbb{N}$ for all $i \in \mathbb{N}_1^m$. Let $T, T_1, T_2, T_3 : \mathcal{T}$ Let v : `<lower-var>`, e : `<exp>`, t : `<type>`. Let v_i : `<lower-var>` for all $i \in \mathbb{N}_1^n$. Let s : `<statement>`. Let c : `<lower-var>` and c_i : `<lower-var>` for all $i \in \mathbb{N}_1^m$. Let $t_{i,j}$: `<type>` for all $i \in \mathbb{N}_1^m$ and $j \in \mathbb{N}_1^{k_i}$. Let Γ be a type context and Δ a variable context. Let A be a set and $[\![.]\!]$: values(`<statement>`)$\cup$values(`<expression>`) $\to$ A.

—

The typing rules for statements are defined in table 3.3.1.

Table 3.3.1: Typing rules for statements

$$\frac{(v,\_) \notin \Delta \quad \Gamma,\Delta \vdash [\![e]\!] : T_1 \quad \Gamma,\Delta \cup \{(v,T_1) \vdash [\![s]\!] : T_2\}}{\Delta,\Gamma \vdash [\![v \texttt{ "=" } e \texttt{ ";" } s]\!] : T_2} \quad [TConstant]$$

$$\frac{(v,\_) \notin \Delta \quad \Gamma,\Delta \vdash [\![e]\!] : T_1 \quad \Gamma,\Delta \cup \{(v,T_1) \vdash [\![s]\!] : T_2\} \quad [\![t]\!]_\Gamma = T_1}{\Gamma,\Delta \vdash [\![v \texttt{ ":" } t \texttt{ ";" } v \texttt{ "=" } e \texttt{ ";" } s]\!] : T_2} \quad [TConstant2]$$

$$\frac{\begin{array}{c}(c,\_) \notin \Gamma \quad (c,\_) \notin \Delta \quad [\![\texttt{t}]\!]_\Gamma = T_1 \\ T_2 \text{ is a mono type} \qquad \{v_1 \dots v_n\} = \text{free}(T_2) \\ \forall v_1 \dots . \forall v_n . T_2 = T_1 \quad \Gamma \cup \{(c,(T_1))\}, \Delta \cup \{(c,T_1)\} \vdash [\![\texttt{s}]\!] : T_3\end{array}}{\Gamma\Delta \vdash [\![\texttt{"type alias" } c\ v_1 \dots v_n \texttt{ "=" } t \texttt{ ";" } s]\!] : T_3} \quad [TAlias]$$

$$\frac{\begin{array}{c}(c,\_) \notin \Gamma \quad (c,\_) \notin \Delta \quad \{v_1 \dots v_n\} = \text{free}(T_2) \\ \mu C.c_1\ [\![t_{1,1}]\!]_\Gamma\ \dots\ [\![t_{1,k_1}]\!]_\Gamma \mid \dots \mid c_m\ [\![t_{m,1}]\!]_\Gamma\ \dots\ [\![t_{m,k_m}]\!]_\Gamma = T_2 \\ \forall v_1 \dots . \forall v_n . T_2 = T_1 \quad \Gamma \cup \{(c,(T_1))\}, \Delta \cup \{(c,T_1)\} \vdash [\![\texttt{s}]\!] : T_3\end{array}}{\Gamma,\Delta \vdash \left[\!\!\left[\begin{array}{l}\texttt{"type" } c\, v_1 \dots v_n \texttt{"="} \\ c_1\ t_{1,1} \dots t_{1,k_1} \mid \dots \mid c_m\ t_{m,1} \dots t_{m,k_m} \texttt{";"} s\end{array}\right]\!\!\right] : T_3} \quad [TCustomType]$$

$$\frac{\Gamma,\Delta \vdash [\![e]\!] : T}{\Gamma,\Delta \vdash [\![\texttt{"main" } e]\!] : T} \quad [TMain]$$

$$\frac{\Gamma,\Delta \vdash [\![e]\!] : T \quad [\![t]\!]_\Gamma = T}{\Gamma,\Delta \vdash [\![\texttt{"main :" } t \texttt{ "; main" } e]\!] : T} \quad [TMain2]$$

TConstant,TConstant2 Check if v is still free then add (v, T_1) to the variable context and evaluate the next statement.

TAlias Check if c is still free. $\{v_1, \dots, v_2\}$ needs to be the set of all free variables in T_2. If all checks are valid we add (v, T_1) to the type context and evaluate the next statement.

TCustomType Similar to [TAlias] we add (v, T_1) to the type context with the only difference that we explicitly define T_1 as an algebraic type.

TMain,TMain2 Evaluate e.

3.3.2 Typing rules for expressions

For the typing rules of expressions we will need to introduce a pattern matching function:

$$\text{match}_\Theta : \text{value}(\texttt{<type>}) \times \text{value}(\texttt{<exp>}) \to \{\mathit{True}, \mathit{False}\}$$

for a given substitution Θ.

The function will be defined afterwards. For now its definition will be arbitrary.

Definition 3.6: type inference for expressions

Let $n \in \mathbb{N}$. Let $T, T_0, T_1, T_2, T_3 : \mathcal{T}$. Let v, v_0, v_1, v_2, v_3 : <lower-var>. Let e, e_0, e_1, e_2, e_3 : <exp>. Let $T_i : \mathcal{T}$ for all $i \in \mathbb{N}_4^n$. Let v_i : <lower-var> for $i \in \mathbb{N}_4^n$. Let $a_i : \mathcal{V}$ for all $i \in \mathbb{N}_4^n$. Let e_i : <exp> for all $i \in \mathbb{N}_4^n$. Let p : <pattern>, le : <list.exp>, lc : <list-case>, lef : <list-exp-field>, c : <upper-var> and t : <type>. Let Γ be a type context and Δ a variable context. Let A be a set and $[\![.]\!]$: values(<statement>) $\cup$ values(<expression>) $\to A$.

—

The typing rules for expressions can be found in table 3.3.2.

Table 3.3.2: Typing rules for expressions

$$\frac{(v, T) \in \Delta}{\Gamma, \Delta \vdash [\![v]\!] : T} \quad [\mathit{TVariable}]$$

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

$$\frac{[\![e_1]\!]_\Gamma : T_1 \quad [\![e_2]\!]_\Gamma : T_2}{\Gamma, \Delta \vdash [\![\texttt{"("}\ e_1\ \texttt{","}\ e_2\ \texttt{")"}]\!] : \{1 : T_1, 2 : T_2\}} \quad [\mathit{TTuple}]$$

$$\Gamma, \Delta \vdash [\![\texttt{"[]"}]\!] : \forall a. \mathit{List}\ a \quad [\mathit{TEmptyList}]$$

$$\frac{\Gamma, \Delta \vdash [\![e]\!] : T}{\Gamma, \Delta \vdash [\![\texttt{"["}\ e\ \texttt{"]"}]\!] : \mathit{List}\ T} \quad [\mathit{TSingleList}]$$

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

$$\frac{e : \texttt{<int>}}{\Gamma, \Delta \vdash [\![e]\!] : \mathit{Int}} \quad [\mathit{TInt}]$$

$$\frac{e : \texttt{<bool>}}{\Gamma, \Delta \vdash [\![e]\!] : \mathit{Bool}} \quad [\mathit{TBool}]$$

$$\frac{\Gamma, \Delta \vdash [\![e_1]\!] : T_1 \to T_2 \quad \Gamma, \Delta \vdash [\![e_2]\!] : T_1}{\Gamma, \Delta \vdash [\![e_1\ e_2]\!] : T_2} \quad [\mathit{TCall}]$$

$$\frac{\Gamma, \Delta \vdash [\![e_1]\!] : T_1 \quad \Gamma, \Delta \vdash \text{match}_\Theta(T_1, [\![p]\!]) \quad \Gamma, \Delta \cup \Theta \vdash [\![e_2]\!] : T_2}{\Gamma, \Delta \vdash [\![\texttt{"case"}\ e_1\ \texttt{"of"}\ \texttt{"["}\ p\ \texttt{"->"}\ e_2\ \texttt{"]"}]\!] : T_2} \quad [\mathit{TSingleCaseOf}]$$

$$\frac{\begin{array}{c}\Gamma, \Delta \vdash [\![e_1]\!] : T_1 \quad \Gamma, \Delta \vdash \text{match}_\Theta(T_1, [\![p]\!]) \quad \Gamma, \Delta \cup \Theta \vdash [\![e_2]\!] : T_2 \\ \Gamma, \Delta \vdash [\![\texttt{"case"}\ e_1\ \texttt{"of"}\ \texttt{"["}\ lc\ \texttt{"]"}]\!] : T_2\end{array}}{\Gamma, \Delta \vdash [\![\texttt{"case"}\ e_1\ \texttt{"of"}\ \texttt{"["}\ p\ \texttt{"->"}\ e_2\ \texttt{";"}\ lc\ \texttt{"]"}]\!] : T_2} \quad [\mathit{TCaseOf}]$$

$$\frac{(v, \_) \notin \Delta \quad \Gamma, \Delta \vdash [\![e_1]\!] : T_1 \quad \Gamma, \Delta \cup \{(v, T_1)\} \vdash [\![e_2]\!] : T_2}{\Gamma, \Delta \vdash [\![\texttt{"let"}\ v\ \texttt{"="}\ e_1\ \texttt{"in"}\ e_2]\!] : T_2} \quad [\mathit{TLetIn}]$$

$$\frac{(v,\_) \notin \Delta \quad [\![t]\!]_\Gamma = T_1 \quad \Gamma,\Delta \vdash [\![e_1]\!] : T_1 \quad \Gamma, \Delta \cup \{(v,T_1)\} \vdash [\![e_2]\!] : T_2}{\Gamma,\Delta \vdash [\![\texttt{"let"}\ v\ \texttt{":"}\ t\ \texttt{";"}\ v\ \texttt{"="}\ e_1\ \texttt{"in"}\ e_2]\!] : T_2} \qquad [\mathit{TLetIn2}]$$

$$\frac{(v_1, \{v_2 : T, \dots\}) \in \Delta}{\Gamma,\Delta \vdash [\![v_1\ \texttt{"."}\ v_2]\!] : T} \qquad [\mathit{TGetter}]$$

$$\frac{\begin{array}{c}\Gamma,\Delta \vdash [\![e]\!] : T_2 \\ T_1 = \{v_2 : T_2, \dots\} \quad (v_1, T_1) \in \Delta\end{array}}{\Gamma,\Delta \vdash [\![\texttt{"\{"}\ v1\ \texttt{"|"}\ v2\ \texttt{"="}\ e\ \texttt{"\}"}]\!] : T_1} \qquad [\mathit{TSingleSetter}]$$

$$\frac{\begin{array}{c}\mathit{lef} : \texttt{<list-exp-field>} \quad \Gamma,\Delta \vdash [\![e]\!] : T_2 \\ T_1 = \{v_2 : T_2, ..\} \quad (v_1, T_1) \in \Delta \quad \Gamma,\Delta \vdash [\![\texttt{"\{"}\ v\ \texttt{"|"}\ \mathit{lef}\ \texttt{"\}"}]\!] : T_1\end{array}}{\Gamma,\Delta \vdash [\![\texttt{"\{"}\ v\ \texttt{"|"}\ v2\ \texttt{"="}\ e\ \texttt{","}\ \mathit{lef}\ \texttt{"\}"}]\!] : T_1} \qquad [\mathit{TSetter}]$$

$$\vdash [\![\texttt{"\{\}"}]\!] : \{\} \qquad [\mathit{TEmptyRecord}]$$

$$\frac{\Gamma,\Delta \vdash \forall i \in \mathbb{N}_1^n . [\![e_i]\!] : T_i}{\Gamma,\Delta \vdash [\![\texttt{"\{"}\ v_1\ \texttt{"="}\ e_1\ \texttt{","}\ \texttt{…","}\ v_n\ \texttt{"="}\ e_n\ \texttt{"\}"}]\!] : \{v_1 : T_1, \dots, v_n : T_n\}} \qquad [\mathit{TRecord}]$$

$$\frac{\Gamma,\Delta \vdash [\![e_1]\!] : \mathit{Bool} \quad \Gamma,\Delta \vdash [\![e_2]\!] : T \quad \Gamma,\Delta \vdash [\![e_3]\!] : T}{\Gamma,\Delta \vdash [\![\texttt{"if"}\ e_1\ \texttt{"then"}\ e_2\ \texttt{"else"}\ e_3]\!] : T} \qquad [\mathit{TIfElse}]$$

$$\frac{\Gamma,\Delta \vdash [\![e_1]\!] : T_1 \to T_2 \quad \Gamma,\Delta \vdash [\![e_2]\!] : T_2 \to T_3}{\Gamma,\Delta \vdash [\![e_1\ \texttt{">>"}\ e_2]\!] : T_1 \to T_3} \qquad [\mathit{TComposition}]$$

$$\frac{\Gamma,\Delta \vdash [\![e_1]\!] : T_1 \quad \Gamma,\Delta \vdash [\![e_2]\!] : T_1 \to T_2}{\Gamma,\Delta \vdash [\![e_1\ \texttt{"|>"}\ e_2]\!] : T_2} \qquad [\mathit{TPipe}]$$

$$\vdash [\![\texttt{"(||)"}]\!] : \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool} \qquad [\mathit{TOr}]$$

$$\vdash [\![\texttt{"(\&\&)"}]\!] : \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool} \qquad [\mathit{TAnd}]$$

$$\vdash [\![\texttt{"not"}]\!] : \mathit{Bool} \to \mathit{Bool} \qquad [\mathit{TNot}]$$

$$\vdash [\![\texttt{"(==)"}]\!] : \mathit{Int} \to \mathit{Int} \to \mathit{Bool} \qquad [\mathit{TEqual}]$$

$$\vdash [\![\texttt{"(<)"}]\!] : \mathit{Int} \to \mathit{Int} \to \mathit{Bool} \qquad [\mathit{TLess}]$$

$$\vdash [\![\texttt{"(//)"}]\!] : \mathit{Int} \to \mathit{Int} \to \mathit{int} \qquad [\mathit{TDivide}]$$

$$\vdash [\![\texttt{"(*)"}]\!] : \mathit{Int} \to \mathit{Int} \to \mathit{int} \qquad [\mathit{TMultiply}]$$

$$\vdash [\![\texttt{"(-)"}]\!] : \mathit{Int} \to \mathit{Int} \to \mathit{int} \qquad [\mathit{TMinus}]$$

$$\vdash [\![\texttt{"(+)"}]\!] : \mathit{Int} \to \mathit{Int} \to \mathit{int} \qquad [\mathit{TPlus}]$$

$$\vdash [\![\texttt{"(::)"}]\!] : \forall a. a \to \mathit{List}\ a \to \mathit{List}\ a \qquad [\mathit{TCons}]$$

$$\vdash [\![\texttt{"foldl"}]\!] : \forall a. \forall b. (a \to b \to b) \to b \to \mathit{List}\ a \to b \qquad [\mathit{TFoldl}]$$

TVariable Find the type in the context.

TLambda Elm allows the parameters of a function to be pattern matched. Therefore, we first need to find a matching type T_1 and can then infer the type of e by including the additional bindings Θ to the context.

TTuple Find the types of e_1 and e_2, then construct the tuple.

TEmptyList The empty list is a literal for every list, therefore we can infer the list poly type.

TSingleList, TList Recursively we check that every element has the same type.

TInt, TBool The type of literals can be inferred without any restrictions.

TCall The first expression needs to be a function that the the second type can be passed to.

TSingleCaseOf, TCaseOf First match the type of the expression e_1 to the pattern, then use the additional bindings Θ to obtain the type of e_2. As all patterns need to have the same type, we can then recursively check the other patterns as well.

TLetIn, TLetIn2 The variable v may not have a value assined in the conext Γ. If so, we can infer the type T_1 of e_1 and add (v, T_1) to the context before we evaluate e_2. For $[TLetIn2]$ we already the type is already given as t. Note that t can be more specific as the type we would usually infer.

TGetter The second variable v_2 is a label of the record, that is bound to v_1.

TSingleSetter, TSetter Setters can not change the type in Elm. But we still need to ensure that the fields are also correctly typed.

TElmptyRecord The empty record can be directly infered, as it has only one element.

TRecord Each field and its value must be given at the same time. That is why we can not use a recursive definition.

TIfElse The first expression e_1 needs to be a boolean and the branches e_2, e_3 must have the same type.

TComposition, TPipe The pipe applies the first expression to the second. The composition is similar to the pipe, but results in a function.

TOr, TAnd, TNot, TEqual, TDivide, TMultiply, TMinus, TPlus, TCons, TFoldl These functions can be seen as lambda function literals.

Example 3.1

In example **??** we have looked at the syntax for a list reversing function. We can now prove the typing of the `reverse` function for $\Gamma = \varnothing$, $\Delta = \varnothing$ and $T = \forall a.List\ a \rightarrow List\ a$.

```
reverse : List a -> List a
reverse =
  foldl (::) []
```

Let $T_1 = List\ a$, $T_0 = List\ a \rightarrow List\ a$ and $T_2 = a \rightarrow List\ a \rightarrow List\ a$

$$\dfrac{\dfrac{\dfrac{\top}{\vdash [\![\texttt{"foldl"}]\!] : \forall a.T_2 \to T_1 \to T_0}\,(4) \quad \dfrac{\top}{\vdash [\![\texttt{"(::)"}]\!] : T_2}\,(3)}{\vdash [\![\texttt{"foldl (::)"}]\!] : \forall a.T_1 \to T_0}\,(1) \quad \dfrac{\top}{\vdash [\![\texttt{"[]"}]\!] : \forall a.T_1}\,(2)}{\vdash [\![\texttt{"(foldl (::)) []"}]\!] : \forall a.T_0}\,(1)$$

$(1)[\mathit{TCall}], (2)[\mathit{TEmptyList}], (3)[\mathit{TCons}], (4)[\mathit{TFoldl}]$

References

[Pie+02] B.C. Pierce et al. *Types and Programming Languages*. The MIT Press. MIT Press, 2002. ISBN: 9780262162098. URL: https://books.google.at/books?id=ti6zoAC9Ph8C.