

### 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  $\Gamma$  be a type context. Let  $e$  be arbitrary.

—

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

$$e \in \text{values}_{\Gamma}(T)$$

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

#### Definition 3.2: Semantics of $\langle \text{type} \rangle$

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

—

We define

$$\begin{aligned} \llbracket \cdot \rrbracket_{\Gamma} : \text{value}(\langle \text{type} \rangle) &\rightarrow \mathcal{T} \\ \llbracket \text{Bool} \rrbracket_{\Gamma} &= \mu \_. \text{True} \mid \text{False} \\ \llbracket \text{Int} \rrbracket_{\Gamma} &= \mu \_. 0 \mid \text{Pos Nat} \mid \text{Neg Nat} \\ \llbracket \text{List} \rrbracket_{\Gamma} &= \forall a. \mu C. [] \mid \text{Cons } a \ C \\ \llbracket "(" \ t_1 \ , \ t_2 \ ")" \rrbracket_{\Gamma} &= \{1 : \llbracket t_1 \rrbracket_{\Gamma}, 2 : \llbracket t_2 \rrbracket_{\Gamma}\} \\ \llbracket "\{ \}" \rrbracket_{\Gamma} &= \{\} \\ \llbracket "\{ " \ v_1 \ " : " \ t_1 \ " , " \ \dots \ " , " \ v_n \ " : " \ t_n \ " \}" \rrbracket_{\Gamma} &= \{v_1 : \llbracket t_1 \rrbracket_{\Gamma}, \dots, v_n : \llbracket t_n \rrbracket_{\Gamma}\} \\ \llbracket t_1 \ " \rightarrow " \ t_2 \rrbracket_{\Gamma} &= \llbracket t_1 \rrbracket_{\Gamma} \rightarrow \llbracket t_2 \rrbracket_{\Gamma} \\ \llbracket c \ t_1 \dots t_n \rrbracket_{\Gamma} &= \llbracket [c] \rrbracket_{\Gamma}(a_1, \dots, a_n) \end{aligned}$$

We say  $\llbracket \cdot \rrbracket_{\Gamma}$  is the *interpretation* of  $\langle \text{type} \rangle$ . For any  $t \in \langle \text{type} \rangle$ , we call  $\llbracket t \rrbracket_{\Gamma}$  the *semantic* of  $t$ .

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

$$\llbracket \cdot \rrbracket : \text{values}(\langle \text{statement} \rangle) \cup \text{values}(\langle \text{expression} \rangle) \rightarrow A$$

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

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

Types are typically inferred using so called *inference rules* [Pie+02]. Using these rules we can construct a truth tree, whose leaves are axioms, to prove that  $\Delta \vdash \llbracket e \rrbracket : T$  for an expression  $e$ , a type  $T$ , a variable context  $\Delta$  and an interpretation function for expressions  $\Delta$ . Such a proving process is called *type checking*. It is also possible to find the most general  $T$  such that  $\Delta \vdash \llbracket e \rrbracket : T$  using the inference rules. This is called *type inference*.

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

### Definition 3.4: Instantiation, Generalization

Let  $T', T \in \mathcal{T}$  and  $e \in \text{values}(\langle \text{statement} \rangle) \cup \text{values}(\langle \text{expression} \rangle)$ . Let  $a$  be a type variable. Let  $\Delta$  be a variable context. Let  $A$  be a set and  $\llbracket . \rrbracket : \text{values}(\langle \text{statement} \rangle) \cup \text{values}(\langle \text{expression} \rangle) \rightarrow A$

$$\frac{T' \sqsubseteq T \quad \Gamma, \Delta \vdash \llbracket e \rrbracket : T'}{\Gamma, \Delta \vdash \llbracket e \rrbracket : T} \quad [\textit{Instantiation}]$$

$$\frac{(a, \_) \notin \Delta \quad \Gamma, \Delta \vdash \llbracket e \rrbracket : T}{\Gamma, \Delta \vdash \llbracket e \rrbracket : \forall a. T} \quad [\textit{Generalization}]$$

The *[Instantiation]* rule says that if a type can be inferred, the same holds for a more specific type. The *[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 Type inference for statement

The inference rules for statements are as follows.

### Definition 3.5: type inference for statement

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  $v : \langle \text{lower-var} \rangle, e : \langle \text{exp} \rangle, t : \langle \text{type} \rangle$ . Let  $v_i : \langle \text{lower-var} \rangle$  for all  $i \in \mathbb{N}_1^n$ . Let  $s : \langle \text{statement} \rangle$ . Let  $c : \langle \text{lower-var} \rangle$  and  $c_i : \langle \text{lower-var} \rangle$  for all  $i \in \mathbb{N}_1^m$ . Let  $t_{i,j} : \langle \text{type} \rangle$  for all  $i \in \mathbb{N}_1^m$  and  $j \in \mathbb{N}_1^{k_i}$ . Let  $\Gamma$  be a type context and  $\Delta$  a variable context. Let  $A$  be a set and  $\llbracket . \rrbracket : \text{values}(\langle \text{statement} \rangle) \cup \text{values}(\langle \text{expression} \rangle) \rightarrow A$ .

The inference rules for statements are defined in table 3.3.1.

Table 3.3.1: Inference rule for statements

---


$$\begin{array}{c}
\frac{(v, \_) \notin \Delta \quad \Gamma, \Delta \vdash \llbracket e \rrbracket : T_1 \quad \Gamma, \Delta \cup \{(v, T_1) \vdash \llbracket s \rrbracket : T_2\}}{\Delta, \Gamma \vdash \llbracket v \text{ "=" } e \text{ ";" } s \rrbracket : T_2} \quad [TConstant] \\
\\
\frac{(v, \_) \notin \Delta \quad \Gamma, \Delta \vdash \llbracket e \rrbracket : T_1 \quad \Gamma, \Delta \cup \{(v, T_1) \vdash \llbracket s \rrbracket : T_2\} \quad \llbracket t \rrbracket_\Gamma = T_1}{\Gamma, \Delta \vdash \llbracket v \text{ ":" } t \text{ ";" } v \text{ "=" } e \text{ ";" } s \rrbracket : T_2} \quad [TConstant2] \\
\\
\frac{\begin{array}{c} (c, \_) \notin \Gamma \quad (c, \_) \notin \Delta \quad \llbracket t \rrbracket_\Gamma = T_1 \\ T_2 \text{ is a mono type} \quad \{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 \llbracket s \rrbracket : T_3 \end{array}}{\Gamma, \Delta \vdash \llbracket \text{"type alias" } c \text{ } v_1 \dots v_n \text{ "=" } t \text{ ";" } s \rrbracket : 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 \llbracket t_{1,1} \rrbracket_\Gamma \dots \llbracket t_{1,k_1} \rrbracket_\Gamma \mid \dots \mid c_m \llbracket t_{m,1} \rrbracket_\Gamma \dots \llbracket t_{m,k_m} \rrbracket_\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 \llbracket s \rrbracket : T_3 \end{array}}{\Gamma, \Delta \vdash \left\llbracket \begin{array}{l} \text{"type" } c \text{ } v_1 \dots v_n \text{ "="} \\ c_1 \text{ } t_{1,1} \dots t_{1,k_1} \mid \dots \mid c_m \text{ } t_{m,1} \dots t_{m,k_m} \text{ ";" } s \end{array} \right\rrbracket : T_3} \quad [TCustomType] \\
\\
\frac{\Gamma, \Delta \vdash \llbracket e \rrbracket : T}{\Gamma, \Delta \vdash \llbracket \text{"main" } e \rrbracket : T} \quad [TMain] \\
\\
\frac{\Gamma, \Delta \vdash \llbracket e \rrbracket : T \quad \llbracket t \rrbracket_\Gamma = T}{\Gamma, \Delta \vdash \llbracket \text{"main" : " } t \text{ ";" main" } e \rrbracket : T} \quad [TMain2]
\end{array}$$


---

**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.  $\{v_1, \dots, v_n\}$  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 Type inference for expressions

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

$$\text{match}_\Theta : \text{value}(\langle \text{type} \rangle) \times \text{value}(\langle \text{exp} \rangle) \rightarrow \{True, False\}$$

for a given substitution  $\Theta$ .

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  $v, v_0, v_1, v_2, v_3 : \langle \text{lower-var} \rangle$ . Let  $e, e_0, e_1, e_2, e_3 : \langle \text{exp} \rangle$ . Let  $T_i : \mathcal{T}$  for all  $i \in \mathbb{N}_4^n$ . Let  $v_i : \langle \text{lower-var} \rangle$  for  $i \in \mathbb{N}_4^n$ . Let  $a_i : \mathcal{V}$  for all  $i \in \mathbb{N}_4^n$ . Let  $e_i : \langle \text{exp} \rangle$  for all  $i \in \mathbb{N}_4^n$ . Let  $p : \langle \text{pattern} \rangle$ ,  $le : \langle \text{list.exp} \rangle$ ,  $lc : \langle \text{list-case} \rangle$ ,  $lef : \langle \text{list-exp-field} \rangle$ ,  $c : \langle \text{upper-var} \rangle$  and  $t : \langle \text{type} \rangle$ . Let  $\Gamma$  be a type context and  $\Delta$  a variable context. Let  $A$  be a set and  $\llbracket \cdot \rrbracket : \text{values}(\langle \text{statement} \rangle) \cup \text{values}(\langle \text{expression} \rangle) \rightarrow A$ .

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

Table 3.3.2: Inference rule for expressions

$\frac{(v, T) \in \Delta}{\Gamma, \Delta \vdash \llbracket v \rrbracket : T}$	$[TVariable]$
$\frac{\Gamma, \Delta \vdash \text{match}_{\Theta}(T_1, p) \quad \Gamma, \Delta \cup \Theta \vdash \llbracket e \rrbracket : T_2}{\Gamma, \Delta \vdash \llbracket "\backslash" p "->" e \rrbracket : T_1 \rightarrow T_2}$	$[TLambda]$
$\frac{\llbracket e_1 \rrbracket_{\Gamma} : T_1 \quad \llbracket e_2 \rrbracket_{\Gamma} : T_2}{\Gamma, \Delta \vdash \llbracket "(" e_1 ", " e_2 ")" \rrbracket : \{1 : T_1, 2 : T_2\}}$	$[TTuple]$
$\Gamma, \Delta \vdash \llbracket "[]" \rrbracket : \forall a. List\ a$	$[TEmptyList]$
$\frac{\Gamma, \Delta \vdash \llbracket e \rrbracket : T}{\Gamma, \Delta \vdash \llbracket "[" e "]" \rrbracket : List\ T}$	$[TSingleList]$
$\frac{\Gamma, \Delta \vdash \llbracket e \rrbracket : T \quad \Gamma, \Delta \vdash \llbracket "[" le "]" \rrbracket : List\ T}{\Gamma, \Delta \vdash \llbracket "[" e ", " le "]" \rrbracket : List\ T}$	$[TList]$
$\frac{e : \langle \text{int} \rangle}{\Gamma, \Delta \vdash \llbracket e \rrbracket : Int}$	$[TInt]$
$\frac{e : \langle \text{bool} \rangle}{\Gamma, \Delta \vdash \llbracket e \rrbracket : Bool}$	$[TBool]$
$\frac{\Gamma, \Delta \vdash \llbracket e_1 \rrbracket : T_1 \rightarrow T_2 \quad \Gamma, \Delta \vdash \llbracket e_2 \rrbracket : T_1}{\Gamma, \Delta \vdash \llbracket e_1\ e_2 \rrbracket : T_2}$	$[TCall]$
$\frac{\Gamma, \Delta \vdash \llbracket e_1 \rrbracket : T_1 \quad \Gamma, \Delta \vdash \text{match}_{\Theta}(T_1, \llbracket p \rrbracket) \quad \Gamma, \Delta \cup \Theta \vdash \llbracket e_2 \rrbracket : T_2}{\Gamma, \Delta \vdash \llbracket "\text{case}" e_1 "\text{of}" "[" p "->" e_2 "]" \rrbracket : T_2}$	$[TSingleCaseOf]$
$\frac{\Gamma, \Delta \vdash \llbracket e_1 \rrbracket : T_1 \quad \Gamma, \Delta \vdash \text{match}_{\Theta}(T_1, \llbracket p \rrbracket) \quad \Gamma, \Delta \cup \Theta \vdash \llbracket e_2 \rrbracket : T_2 \quad \Gamma, \Delta \vdash \llbracket "\text{case}" e_1 "\text{of}" "[" lc "]" \rrbracket : T_2}{\Gamma, \Delta \vdash \llbracket "\text{case}" e_1 "\text{of}" "[" p "->" e_2 "; " lc "]" \rrbracket : T_2}$	$[TCaseOf]$
$\frac{(v, \_) \notin \Delta \quad \Gamma, \Delta \vdash \llbracket e_1 \rrbracket : T_1 \quad \Gamma, \Delta \cup \{(v, T_1)\} \vdash \llbracket e_2 \rrbracket : T_2}{\Gamma, \Delta \vdash \llbracket "\text{let}" v "=" e_1 "\text{in}" e_2 \rrbracket : T_2}$	$[TLetIn]$

$\frac{(v, \_) \notin \Delta \quad \llbracket t \rrbracket_{\Gamma} = T_1 \quad \Gamma, \Delta \vdash \llbracket e_1 \rrbracket : T_1 \quad \Gamma, \Delta \cup \{(v, T_1)\} \vdash \llbracket e_2 \rrbracket : T_2}{\Gamma, \Delta \vdash \llbracket \text{"let" } v \text{" : " } t \text{" ; " } v \text{" = " } e_1 \text{" in" } e_2 \rrbracket : T_2}$	$[TLetIn2]$
$\frac{(v_1, \{v_2 : T, \dots\}) \in \Delta}{\Gamma, \Delta \vdash \llbracket v_1 \text{" . " } v_2 \rrbracket : T}$	$[TGetter]$
$\frac{\Gamma, \Delta \vdash \llbracket e \rrbracket : T_2 \quad T_1 = \{v_2 : T_2, \dots\} \quad (v_1, T_1) \in \Delta}{\Gamma, \Delta \vdash \llbracket \text{"{" } v_1 \text{"   " } v_2 \text{" = " } e \text{" } \rrbracket : T_1}$	$[TSingleSetter]$
$\frac{lef : \langle \text{list-exp-field} \rangle \quad \Gamma, \Delta \vdash \llbracket e \rrbracket : T_2 \quad T_1 = \{v_2 : T_2, ..\} \quad (v_1, T_1) \in \Delta \quad \Gamma, \Delta \vdash \llbracket \text{"{" } v \text{"   " } lef \text{" } \rrbracket : T_1}{\Gamma, \Delta \vdash \llbracket \text{"{" } v \text{"   " } v_2 \text{" = " } e \text{" , " } lef \text{" } \rrbracket : T_1}$	$[TSetter]$
$\vdash \llbracket \text{"{" } \rrbracket : \{\}$	$[TEmptyRecord]$
$\frac{\Gamma, \Delta \vdash \forall i \in \mathbb{N}_1^n. \llbracket e_i \rrbracket : T_i}{\Gamma, \Delta \vdash \llbracket \text{"{" } v_1 \text{" = " } e_1 \text{" , " } \dots \text{" , " } v_n \text{" = " } e_n \text{" } \rrbracket : \{v_1 : T_1, \dots, v_n : T_n\}}$	$[TRecord]$
$\frac{\Gamma, \Delta \vdash \llbracket e_1 \rrbracket : Bool \quad \Gamma, \Delta \vdash \llbracket e_2 \rrbracket : T \quad \Gamma, \Delta \vdash \llbracket e_3 \rrbracket : T}{\Gamma, \Delta \vdash \llbracket \text{"if" } e_1 \text{" then" } e_2 \text{" else" } e_3 \rrbracket : T}$	$[TIfElse]$
$\frac{\Gamma, \Delta \vdash \llbracket e_1 \rrbracket : T_1 \rightarrow T_2 \quad \Gamma, \Delta \vdash \llbracket e_2 \rrbracket : T_2 \rightarrow T_3}{\Gamma, \Delta \vdash \llbracket e_1 \text{" >> " } e_2 \rrbracket : T_1 \rightarrow T_3}$	$[TComposition]$
$\frac{\Gamma, \Delta \vdash \llbracket e_1 \rrbracket : T_1 \quad \Gamma, \Delta \vdash \llbracket e_2 \rrbracket : T_1 \rightarrow T_2}{\Gamma, \Delta \vdash \llbracket e_1 \text{"  > " } e_2 \rrbracket : T_2}$	$[TPipe]$
$\vdash \llbracket \text{"(   )" } \rrbracket : Bool \rightarrow Bool \rightarrow Bool$	$[TOr]$
$\vdash \llbracket \text{"(&&)" } \rrbracket : Bool \rightarrow Bool \rightarrow Bool$	$[TAnd]$
$\vdash \llbracket \text{"not" } \rrbracket : Bool \rightarrow Bool$	$[TNot]$
$\vdash \llbracket \text{"(==)" } \rrbracket : Int \rightarrow Int \rightarrow Bool$	$[TEqual]$
$\vdash \llbracket \text{"(<)" } \rrbracket : Int \rightarrow Int \rightarrow Bool$	$[TLess]$
$\vdash \llbracket \text{"(/)" } \rrbracket : Int \rightarrow Int \rightarrow int$	$[TDivide]$
$\vdash \llbracket \text{"(*)" } \rrbracket : Int \rightarrow Int \rightarrow int$	$[TMultiply]$
$\vdash \llbracket \text{"(-)" } \rrbracket : Int \rightarrow Int \rightarrow int$	$[TMinus]$
$\vdash \llbracket \text{"(+)" } \rrbracket : Int \rightarrow Int \rightarrow int$	$[TPlus]$
$\vdash \llbracket \text{"(:)" } \rrbracket : \forall a. a \rightarrow List\ a \rightarrow List\ a$	$[TCons]$
$\vdash \llbracket \text{"foldl" } \rrbracket : \forall a. \forall b. (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow List\ a \rightarrow b$	$[TFoldl]$

---

**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  $\Theta$  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  $\Theta$  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  $\Gamma$ . 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.

**TEmptyRecord** 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  $\Gamma = \emptyset, \Delta = \emptyset$  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  $T_2 = a \rightarrow List\ a \rightarrow List\ a$

$$\begin{array}{c}
(4) \frac{\top}{\vdash \llbracket \text{foldl} \rrbracket : \forall a. T_2 \rightarrow T_1 \rightarrow T_0} \quad \frac{\top}{\vdash \llbracket (:) \rrbracket : T_2} \quad (3) \\
(1) \frac{\vdash \llbracket \text{foldl } (:) \rrbracket : \forall a. T_1 \rightarrow T_0 \quad \frac{\top}{\vdash \llbracket [] \rrbracket : \forall a. T_1} (2)}{\vdash \llbracket (\text{foldl } (:) ) [] \rrbracket : \forall a. T_0} (1)
\end{array}$$

(1)[*TCall*], (2)[*TEmptyList*], (3)[*TCons*], (4)[*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>.