

### 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  $\Gamma$  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  $ltf \in \langle \text{list-type-fields} \rangle$  the judgment has the form

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

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

For  $lt \in \langle \text{list-type} \rangle$  the judgment has the form

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

which can be read as “given  $\Gamma$ ,  $lt$  defines the list  $(T_1, \dots, T_n)$ ”.

For  $t \in \langle \text{type} \rangle$  the judgment has the form

$$\Gamma \vdash t : T$$

which can be read as “given  $\Gamma$ ,  $t$  has the type  $T$ ”.

---

#### PATTERN JUDGMENTS

For pattern judgments, let  $\Gamma, \Delta$  and  $\Theta$  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  $lpl \in \langle \text{list-pattern-list} \rangle$  the judgment has the form

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

which can be read as “given  $\Gamma, \Delta$ , we can match  $\text{List } T$  with the pattern  $lpl$  by using the context  $\Theta$ ”.

For  $lps \in \langle \text{list-pattern-sort} \rangle$  the judgment has the form

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

which can be read as “given  $\Gamma$  and  $\Delta$ , we can match  $(T_1, \dots, T_n)$  with the pattern  $lps$  by using the context  $\Theta$ ”.

For  $lpv \in \langle \text{list-pattern-vars} \rangle$  the judgment has the form

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

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

For  $p \in \langle \text{pattern} \rangle$  the judgment has the form

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

which can be read as “given  $\Gamma$  and  $\Delta$ , we can match  $T$  with the pattern  $p$  by using the context  $\Theta$ ”.

---

#### EXPRESSION JUDGMENTS

For expression judgments, let  $\Gamma, \Delta$  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  $lef \in \langle \text{list-exp-field} \rangle$  the judgment has the form

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

which can be read as “given  $\Gamma$  and  $\Delta$ ,  $lef$  has the type  $\{a_1 : T_1, \dots, a_n : T_n\}$ ”.

For  $mes \in \langle \text{maybe-exp-sign} \rangle$  the judgment has the form

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

which can be read as “given  $\Gamma, a$  has the type  $T$  under the assumption  $mes$ ”.

For  $lc \in \langle \text{list-case} \rangle$  the judgment has the form

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

which can be read as “given  $\Gamma$  and  $\Delta$  and a type  $T_1$ ,  $lc$  has the type  $T_2$ ”.

For  $b \in \langle \text{bool} \rangle$  the judgment has the form

$$b : T$$

which can be read as “ $b$  has the type  $T$ ”.

For  $i \in \langle \text{int} \rangle$  the judgment has the form

$$e : T$$

which can be read as “ $i$  has the type  $T$ ”.

For  $le \in \langle \text{list-exp} \rangle$  the judgment has the form

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

which can be read as “given  $\Gamma$  and  $\Delta$ ,  $le$  has the type  $\text{List } T$ ”.

For  $e \in \langle \text{exp} \rangle$  the judgment has the form

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

which can be read as “given  $\Gamma$  and  $\Delta$ ,  $e$  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  $lss \in \langle \text{list-statement-sort} \rangle$  the judgment has the form

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

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  $(T_{i,1}, \dots, T_{i,k_i})$ ”.

For  $lsv \in \langle \text{list-statement-var} \rangle$  the judgment has the form

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

which can be read as “ $lsv$  describes the list  $(a_1, \dots, a_n)$ ”.

For  $ls \in \langle \text{list-statement} \rangle$  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  $\Gamma_1$  to  $\Gamma_2$  and  $\Delta_1$  to  $\Delta_2$ ”.

For  $mss \in \langle \text{maybe-statement-sign} \rangle$  the judgment has the form

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

which can be read as “given  $\Gamma$ ,  $a$  has the type  $T_2$  under the assumption  $mss$ ”.

For  $s \in \langle \text{statement} \rangle$  the judgment has the form

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

which can be read as “the statement  $s$  maps  $\Gamma_1$  to  $\Gamma_2$  and  $\Delta_1$  to  $\Delta_2$ ”.

For  $mms \in \langle \text{maybe-main-sign} \rangle$  the judgment has the form

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

which can be read as “the main function has type  $T$  under the assumption  $mms$ ”.

For  $prog \in \langle \text{program} \rangle$  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 equality of two given types  $T_1$  and  $T_2$ .

We will write  $a_1, \dots, a_n = \text{free}(T)$  to denote all free variables  $a_1, \dots, a_n$  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  $\Delta : \mathcal{V} \rightarrow \mathcal{T}$  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  $\Delta$  is a partial function and therefore  $\Delta(e)$  would only be defined if  $T_0$  exists. If  $T_0$  does not 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  $\Delta_1, \Delta_2$  be type contexts,  $a \in \mathcal{V}$ . Let  $T, T' \in \mathcal{T}$  such that  $T'$  is a mono type  $\forall c_1 \dots \forall c_m T' = T$  for some  $c_i \in \mathcal{V}, i \in \mathbb{N}_0^m$ .

We define

$\text{insert}_{\Delta_1}(\Delta_2) :=$

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

This definition essentially states that all quantified variables of  $T$ , that occur in  $\Delta_2$ , 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

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

### 3.3.3 Inference Rules for Type Signatures

---

#### LIST-TYPE-FIELDS

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

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

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

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

---

#### LIST-TYPE

Judgment:  $\Gamma \vdash \text{lt} : (T_1, \dots, T_n)$

$$\Gamma \vdash "" : ()$$

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

---

**TYPE**

Judgment:  $\Gamma \vdash t : T$

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

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

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

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

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

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

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

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 currending type  $T_0$ . For that, let  $\Gamma = \emptyset$ .

$$\frac{\frac{\frac{\overline{\forall a. a = T_3}}{\emptyset \vdash a : T_3} \quad \overline{List\ T_3 = T_1}}{\emptyset \vdash Lista : T_1} \quad \frac{\frac{\frac{\overline{\forall a. a = T_4}}{\emptyset \vdash a : T_4} \quad \overline{List\ T_4 = T_2}}{\emptyset \vdash Lista : T_2}}{T_1 \rightarrow T_2 = T_0}}{\emptyset \vdash List\ a \rightarrow List\ a : T_0}$$

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

### 3.3.4 Inference Rules for patterns

---

#### LIST-PATTERN-LIST

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

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

$$\frac{\Gamma, \Delta \vdash \text{match}_{\Theta_1}(T, p) \quad \Gamma, \Delta \vdash \text{match}_{\Theta_2}(\text{List } T, lpl) \quad \Theta_1 \cap \Theta_2 = \emptyset \quad \Theta_3 = \text{insert}_{\Theta_1}(\Theta_2)}{\Gamma, \Delta \vdash \text{match}_{\Theta_3}(\text{List } T, p \text{ " " } lpl)}$$

$\Theta_3$  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  $\Theta_1$  of  $p$  and the binding  $\Theta_2$  of  $lpl$  are disjoint.

---

#### LIST-PATTERN-SORT

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

$$\Gamma, \Delta \vdash \text{match}_{\emptyset}(), ""$$

$$\frac{\Gamma, \Delta \vdash \text{match}_{\Theta_1}(T_0, p) \quad \Gamma, \Delta \vdash \text{match}_{\Theta_2}((T_1, \dots, T_n), lps) \quad \Theta_1 \cap \Theta_2 = \emptyset \quad \Theta_3 = \text{insert}_{\Theta_1}(\Theta_2)}{\Gamma, \Delta \vdash \text{match}_{\Theta_3}((T_0, T_1, \dots, T_n), p \text{ } lps)}$$

---

#### LIST-PATTERN-VARS

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

$$"" : ()$$

$$\frac{lpv : (a_1, \dots, a_n)}{a_0 \text{ " " } lpv : (a_0, a_1, \dots, a_n)}$$

---

#### PATTERN

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

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

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

$$\frac{\Gamma, \Delta \vdash \text{match}_{\Theta_1}(T_1, p_1) \quad \Gamma, \Delta \vdash \text{match}_{\Theta_2}(T_2, p_2) \quad \Theta_1 \cap \Theta_2 = \emptyset \quad \text{insert}_{\Theta_1}(\Theta_2) = \Theta_3}{\Gamma, \Delta \vdash \text{match}_{\Theta_3}((T_1, T_2), \text{"("p1", "p2")"})}$$

$$\frac{\Delta(c) \sqsubseteq T_1 \rightarrow \dots \rightarrow T_n \rightarrow T_0 \quad \Gamma, \Delta \vdash \text{match}_{\Theta}((T_1, \dots, T_n), \text{lp}ps)}{\Gamma, \Delta \vdash \text{match}_{\Theta}(T_0, c \text{ lp}ps)}$$

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

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

$$\frac{\text{lp}v = (a_1, \dots, a_n) \quad T = \{a_1 : T_1, \dots, a_n : T_n\} \quad \Delta \cap \{(a_1, \_), \dots, (a_n, \_)\} = \emptyset \quad \Theta = \{(a_1, T_1), \dots, (a_n, T_n)\}}{\Gamma, \Delta \vdash \text{match}_{\Theta}(T, \text{"{"lpv"}"})}$$

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

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

### Example 3.2

In example ?? we have looked at the syntax for list reversing function. We will now find the bindings  $\Theta_0$  for the following pattern used in the reversing function.

**a** :: \_

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

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

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

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



### 3.3.5 Inference Rules for Expressions

---

#### LIST-EXP-FIELD

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

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

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

---

#### MAYBE-EXP-SIGN

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

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

If no argument is given, then we do nothing.

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

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

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

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

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

---

#### BOOL

Judgment:  $b : T$

$$b : Bool$$

---

**INT**

Judgment:  $i : T$

$$i : Int$$

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

---

**LIST-EXP**

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

$$\Gamma, \Delta \vdash "" : \forall a. List\ a$$

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

---

**EXP**

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

$$\Gamma, \Delta \vdash "foldl" : \forall a. \forall b. (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow List\ a \rightarrow b$$

$$\Gamma, \Delta \vdash "(::)" : \forall a. a \rightarrow List\ a \rightarrow List\ a$$

$$\Gamma, \Delta \vdash "(+)" : Int \rightarrow Int \rightarrow int$$

$$\Gamma, \Delta \vdash "(-)" : Int \rightarrow Int \rightarrow int$$

$$\Gamma, \Delta \vdash "(*)" : Int \rightarrow Int \rightarrow int$$

$$\Gamma, \Delta \vdash "(//)" : Int \rightarrow Int \rightarrow int$$

$$\Gamma, \Delta \vdash "<" : Int \rightarrow Int \rightarrow Bool$$

$$\Gamma, \Delta \vdash "(==)" : Int \rightarrow Int \rightarrow Bool$$

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

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

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

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

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

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

$$\frac{\Gamma, \Delta \vdash lef : \{a_1 : T_1, \dots, a_n : T_n\}}{\Gamma, \Delta \vdash \text{"{"lef"}"} : \{a_1 : T_1, \dots, a_n : T_n\}}$$

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

$$\frac{\Gamma, \Delta \vdash lef : \{a_1 : T_1, \dots, a_n : T_n\} \quad \Gamma, \Delta \vdash \Delta(a) \subseteq T_0 \quad T_0 = \{a_1 : T_1, \dots, a_n : T_n, \dots\}}{\Gamma, \Delta \vdash \text{"{" a " | " lef "}" } : T_0}$$

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

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

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

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

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

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

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

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

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

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

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  $T_0 = \forall a. \text{List } a \rightarrow \text{List } a$  of the `reverse` function for  $\Gamma = \Delta = \emptyset$ ,  $\Delta = \emptyset$ . The body of the `reverse` function is as follows:

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

$$\frac{\frac{\frac{\emptyset, \emptyset \vdash \text{"foldl"} : T_2}{\emptyset, \emptyset \vdash \text{"foldl"} (::) : T_1} \quad \frac{\emptyset, \emptyset \vdash "(:)" : \forall a. \text{List } a \rightarrow \text{List } a}{\emptyset, \emptyset \vdash \text{"foldl"} (::) : T_1} \quad \frac{\frac{\emptyset, \emptyset \vdash "[]" : \forall a. \text{List } a}{\emptyset, \emptyset \vdash "[]" : \forall a. \text{List } a}}{\emptyset, \emptyset \vdash \text{"foldl"} (::) [] : T_0}$$

where  $T_1 = \forall a. \text{List } a \rightarrow \text{List } a \rightarrow \text{List } a$  and  $T_2 = \forall a. (\text{List } a \rightarrow \text{List } a) \rightarrow \text{List } a \rightarrow \text{List } a \rightarrow \text{List } a$ .

### 3.3.6 Inference Rules for Statements

#### LIST-STATEMENT-VAR

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

" : ()

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

---

**LIST-STATEMENT-SORT**

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

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

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

---

**LIST-STATEMENT**

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

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

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

---

**MAYBE-STATEMENT-SIGN**

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

$\Gamma, \text{ } \vdash a : T$

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

---

**STATEMENT**

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

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

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

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

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  $\Delta_1$  under the name  $c_i$ . In Elm these constructors are the only constants beginning with an upper-case letter.

---

#### MAYBE-MAIN-SIGN

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

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

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

---

#### PROGRAM

Judgment:  $prog : T$

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