3.3 Type Inference

Let us assume we can define a *interpretation* function

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

where Γ is a type context and A is a non-empty set. We will discuss the definition of such a function as well as the definition of A in the next section.

In this section we are more interested in the judgment $[\![\texttt{e}]\!] \in \text{values}_\Gamma(T)$ for a given type T a type context Γ and an expression e. For now let us assume that the definition of A actually allows this.

3.3.1 Typing Judgments

A judgment can arise from a set of logical inference rules [Pie+02].

Definition 3.1: inference rules

Let $n \in \mathbb{N}$, P_i be sequents for all $i \in \mathbb{N}$. Let C be a sequent.

—

We call

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

an *inference rule*.

If a judgment holds, then there exists a set of logical inference rules that proves it.

We will see that the initial judgment ($[\![\texttt{e}]\!] \in \text{values}_\Gamma(T)$) is dependent on another context, this time for variables instead of types:

Definition 3.2: Variable Context

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

We also introduce a new syntax for saying a value is of some type.

Definition 3.3: type of value

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: $e :_\Gamma T$):$\Leftrightarrow$

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

Is $\Gamma = \varnothing$ then we may write $e : T$ instead of $e :_\Gamma T$.

The judgment in question can now be expressed more generally as

$$\Delta \vdash \llbracket \texttt{e} \rrbracket :_\Gamma T$$

where Γ is a type context, Δ is a variable context, $e \in \text{values}_\Gamma(\texttt{<program>}) \cup \text{values}_\Gamma(\texttt{<expression>})$ and T is a type. Note that we originally assumed $\Delta = \emptyset$. Also, note that the values of `<program>` and `<expression>` are disjoint and therefore we will look at $e : \texttt{<program>}$ and $e : \texttt{<expression>}$ separately.

If the type T is known then we talk about *type checking* else we call the judgment a *type inference*. For inferring a type, the result is not necessary unique, that is why we might only want to find the most general type, meaning a type T_1 such that

$$\underbrace{(\forall T_2 \in \mathcal{T} \wedge T_1 \sqsubseteq T_2 . e :_\Gamma T_2)}_{T_1 \text{ is a infered type}} \wedge \underbrace{(\forall T_2 \in \mathcal{T} \wedge T_2 \sqsubseteq T_1 . \exists T_3 \in \mathcal{T} \wedge T_2 \sqsubseteq T_3 . \neg(e :_\Gamma T_3))}_{T_1 \text{ is sharp}} .$$

3.3.2 Auxiliary Definitions

We will need the semantics of `<type>`, namely a function that maps $\text{values}_\Gamma(\texttt{<type>})$ to $\mathcal{T}$.

Definition 3.4: Semantics of <type>

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

—

We define

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

Additionally, 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.

We can already state two universal inference rules for any Hindley-Milner type system.

Definition 3.5: Instantiation, Generalization

Let $T', T \in \mathcal{T}$ and $e \in \text{values}(\texttt{<program>}) \cup \text{values}(\texttt{<expression>})$. Let a be a type variable. Let Δ be a variable context. Let A be a set and $\llbracket . \rrbracket : \text{values}(\texttt{<program>}) \cup \text{values}(\texttt{<expression>}) \to A$

—

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

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

The $[\textit{Instantiation}]$ rule says that if a type can be inferred, the same holds for a more specific type. The $[\textit{Generalization}]$ 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.

3.3.3 Inference Rules for Programs

The inference rules for programs will done statement for statement. Note that every statement has one rule that can be applied, statement with optional parameters have a rule with and one without the optional parameter.

Definition 3.6: Inference rules for programs

Let $n, m \in \mathbb{N}$. Let $k_i \in \mathbb{N}$ for all $i \in \mathbb{N}_1^m$. Let $T, T_1, T_2, T_3 \in \mathcal{T}$ Let $v : \texttt{<lower-var>}, e : \texttt{<exp>}, t : \texttt{<type>}$. Let $v_i : \texttt{<lower-var>}$ for all $i \in \mathbb{N}_1^n$. Let $s : \texttt{<list-statement>}$. Let $c : \texttt{<lower-var>}$ and $c_i : \texttt{<lower-var>}$ for all $i \in \mathbb{N}_1^m$. Let $t_{i,j} : \texttt{<type>}$ for all $i \in \mathbb{N}_1^m$ and $j \in \mathbb{N}_1^{k_i}$. Let $mt : \texttt{<maybe-main-sign>}$ and $me : \texttt{<exp>}$. Let Γ be a type context and Δ a variable context. Let A be a set and $\llbracket . \rrbracket : \text{values}(\texttt{<program>}) \cup \text{values}(\texttt{<expression>}) \to A$.

—

The inference rules for programs are defined in table 1.

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. $\{v_1, \ldots, v_2\}$ 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 $[\textit{TAlias}]$ 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.

Table 1: Inference rules for programs

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

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

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

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

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

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

3.3.4 Inference Rules for Expressions

In the inference rules $[\mathit{TConstant}]$, $[\mathit{TConstant2}]$ and $[\mathit{Main}]$, $[\mathit{Main2}]$ we used a judgment for expressions. We will now give the corresponding inference rules. As before, each expression has one or two rules depending on optional parameters.

Definition 3.7: Inference rules for expressions

Let $n \in \mathbb{N}$. Let $T, T_0, T_1, T_2, T_3 \in \mathcal{T}$. 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 inference rules for expressions can be found in table 2.

Table 2: Inference rules for expressions

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$\vdash [\![$ `"foldl"` $]\!] : \forall a.\forall b.(a \rightarrow b \rightarrow b) \rightarrow b \rightarrow \mathit{List}\ a \rightarrow b \quad [\mathit{TFoldl}]$

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 $T = \forall a.\mathit{List}\ a \rightarrow \mathit{List}\ a$.

```
reverse : List a -> List a
```

```
reverse =
  foldl (::) []
```

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

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

$(1)[TCall], (2)[TEmptyList], (3)[TCons], (4)[TFoldl]$

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.