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 Γ 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* $\in$ `<list-type-fields>` the judgment has the form

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

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

For *lt* $\in$ `<list-type>` the judgment has the form

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

which can be read as "given Γ, *lt* defines the list $(T_1, \dots, T_n)$".

For $t \in$ `<type>` the judgment has the form

$$\Gamma \vdash t : T$$

which can be read as "given Γ, t has the type T".

EXPRESSION JUDGMENTS

For expression judgments, let Γ, Δ 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* $\in$ `<list-exp-field>` the judgment has the form

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

which can be read as "given Γ and Δ, *lef* has the type $\{a_1 : T_1, \ldots, a_n : T_n\}$".

For *mes* $\in$ `<maybe-exp-sign>` the judgment has the form

$$\Gamma, \mathit{mes} \vdash a : T$$

which can be read as "given Γ, a has the type T under the assumption *mes*".

For $b \in$ `<bool>` the judgment has the form

$$b : T$$

which can be read as "b has the type T".

For $i \in$ `<int>` the judgment has the form

$$e : T$$

which can be read as "i has the type T".

For *le* $\in$ `<list-exp>` the judgment has the form

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

which can be read as "given Γ and Δ, *le* has the type *List* T".

For $e \in$ `<exp>` the judgment has the form

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

which can be read as "given Γ and Δ, 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 *lsv* $\in$ `<list-statement-var` the judgment has the form

$$\mathit{lsv} : (a_1, \ldots, a_n)$$

which can be read as "*lsv* describes the list $(a_1, \ldots, a_n)$".

For *ls* $\in$ `<list-statement>` the judgment has the form

$$\Gamma_1, \Delta_2, \mathit{ls} \vdash \Gamma_2, \Delta_2$$

which can be read as "the list of statements ls maps Γ_1 to Γ_2 and Δ_1 to Δ_2".

For $mss \in$ `<maybe-statement-sign>` the judgment has the form

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

which can be read as "given Γ, a has the type T_2 under the assumption mss".

For $s \in$ `<statement>` the judgment has the form

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

which can be read as "the statement s maps Γ_1 to Γ_2 and Δ_1 to Δ_2".

For $mms \in$ `<maybe-main-sign>` the judgment has the form

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

which can be read as "the main function has type T under the assumtion mms".

For $prog \in$ `<program>` 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 equiality of two given types T_1 and T_2.

We will write $a_1, \ldots, a_n = \text{free}(T)$ to denote all free variables $a_1, \ldots, a_n$ 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 $\Delta : \mathcal{V} \nrightarrow \mathcal{T}$ be a type context, $T \in \mathcal{T}$ and e be an expression.

Then we define

$$e \sqsubseteq_\Delta T :\Leftrightarrow \exists T_0 \in \mathcal{T}.(e, T_0) \in \Delta \wedge T_0 \sqsubseteq T$$

Note that Δ 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: Most General Type

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

—

We define $\overline{\Gamma} : \Gamma \to \mathcal{T}$ as

$$\begin{aligned}\overline{\Gamma}(T) :=& \forall a_1 \ldots \forall a_n . T_0 \\ & \text{such that } \{a_1, \ldots, a_n\} = \text{free}(T') \setminus \{a \mid (a, \_) \in \Gamma\} \\ & \text{where } a_i \in \mathcal{V} \text{ for } i \in \mathbb{N}_0^n \text{ and } T_0 \text{ is the mono type of } T.\end{aligned}$$

We say $\overline{\Gamma}(T)$ is *the most general type* of T.

The most general type ensures that all type variables are bound by either an quantifier or a type alias in the type context Γ. It also ensure that every type variable bound by a quantifier occures in the mono type T_0. The act of replacing types with more general onces, by binding free variables, is called *Generalization* and is also typically found as an inference rule in text books.

PREDEFINED TYPES

Additionally, we define

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

3.3.3 Inference Rules for Type Signatures

LIST-TYPE-FIELDS

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

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

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

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

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

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

TYPE

Judgment: $\Gamma \vdash t : T$

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

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

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

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

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

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

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

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 curresponding type T_0. For that, let $\Gamma = \varnothing$.

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\forall a.a = T_3}}{\varnothing \vdash a : T_3} \quad \cfrac{}{List\ T_3 = T_1}}{\varnothing \vdash Lista : T_1} \quad \cfrac{\cfrac{\cfrac{}{\forall a.a = T_4}}{\varnothing \vdash a : T_4} \quad \cfrac{}{List\ T_4 = T_2}}{\varnothing \vdash Lista : T_2}}{\cfrac{T_1 \to T_2 = T_0}{\varnothing \vdash \texttt{List a -> List a} : T_0}}$$

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

3.3.4 Inference Rules for Expressions

LIST-EXP-FIELD

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

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

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

MAYBE-EXP-SIGN

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

$$\Gamma, \texttt{""} \vdash a : T$$

If no argument is given, then we do nothing.

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

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.

BOOL

Judgment: $b : T$

$$b : Bool$$

INT

Judgment: $i : T$

$$i : Int$$

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

LIST-EXP

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

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

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

EXP

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

$$\Gamma, \Delta \vdash \texttt{"foldl"} : \forall a. \forall b. (a \to b \to b) \to b \to List\ a \to b$$

$$\Gamma, \Delta \vdash \texttt{"(::)"} : \forall a. a \to List\ a \to List\ a$$

$$\Gamma, \Delta \vdash \texttt{"(+)"} : Int \to Int \to Int$$

$$\Gamma, \Delta \vdash \texttt{"(-)"} : Int \to Int \to Int$$

$$\Gamma, \Delta \vdash \texttt{"(*)"} : Int \to Int \to Int$$

$$\Gamma, \Delta \vdash \texttt{"(//)"} : Int \to Int \to Int$$

$$\Gamma, \Delta \vdash \texttt{"(<)"} : Int \to Int \to Bool$$

$$\Gamma, \Delta \vdash \texttt{"(==)"} : Int \to Int \to Bool$$

$$\Gamma, \Delta \vdash \texttt{"not"} : Bool \to Bool$$

$$\Gamma, \Delta \vdash \texttt{"(\&\&)"} : Bool \to Bool \to Bool$$

$$\Gamma, \Delta \vdash \texttt{"(||)"} : Bool \to Bool \to Bool$$

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

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

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

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

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

$$\frac{\Gamma, \Delta \vdash \mathit{lef} : \{a_1 : T_1, \dots, a_n : T_n\} \qquad \Gamma, \Delta \vdash a \sqsubseteq_\Delta T_0 \quad T_0 = \{a_1 : T_1, \dots, a_n : T_n, \dots\}}{\Gamma, \Delta \vdash \texttt{"\{"}\ a\ \texttt{"|"}\ \mathit{lef}\ \texttt{"\}"} : T_0}$$

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

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

$$\frac{(a, \_) \notin \Delta \quad \Gamma, \Delta \vdash e_1 : T_1 \quad \mathit{mes} : T_1 \vdash a : T_1 \qquad \Gamma, \Delta \cup \{(a, \overline{\Gamma}(T_1))\} \vdash e_2 : T_2}{\Gamma, \Delta \vdash \texttt{"let"}\ \mathit{mes}\ a\texttt{"="}\ e_1\ \texttt{"in"}\ e_2 : T_2}$$

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

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

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

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

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

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

$$\frac{\Gamma, \Delta \cup \{(a, \overline{\Gamma}(T_1))\} \vdash e : T_2}{\Gamma, \Delta \vdash \texttt{"\textbackslash"}a\texttt{"->"}e : T_1 \to T_2}$$

$$\frac{c \sqsubseteq_\Delta T}{\Gamma, \Delta \vdash c : T}$$

$$\frac{a \sqsubseteq_\Delta T}{\Gamma, \Delta \vdash a : T}$$

Example 3.2

In example **??** we have looked at the syntax for a list reversing function. We can now check the type $T_0 = \forall a.List\ a \to List\ a$ of the `reverse` function for $\Gamma = \Delta = \varnothing$, $\Delta = \varnothing$. The body of the *reverse* function is as follows:

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

$$\dfrac{\dfrac{\dfrac{}{\varnothing, \varnothing \vdash \texttt{"foldl"} : T_2} \quad \dfrac{}{\varnothing, \varnothing \vdash \texttt{"(::)"} : \forall a.List\ a \to List\ a}}{\varnothing, \varnothing \vdash \texttt{"foldl (::)"} : T_1} \quad \dfrac{\dfrac{}{\varnothing, \varnothing \vdash \texttt{""} : \forall a.a}}{\varnothing, \varnothing \vdash \texttt{"[]"} : \forall a.List\ a}}{\varnothing, \varnothing \vdash \texttt{"foldl (::) []"} : T_0}$$

where $T_1 = \forall a.List\ a \to List\ a \to List\ a$ and $T_2 = \forall a.(List\ a \to List\ a) \to List\ a \to List\ a \to List\ a$.

3.3.5 Inference Rules for Statements

LIST-STATEMENT-VAR

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

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

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

LIST-STATEMENT-SORT

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

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

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

LIST-STATEMENT

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

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

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

MAYBE-STATEMENT-SIGN

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

$$\Gamma, \texttt{""} \vdash a : T$$

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

STATEMENT

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

$$\frac{\begin{array}{c} \Gamma_1 = \Gamma_2 \quad (a, \_) \notin \Delta_1 \\ \Gamma_1, mss \vdash e : T \quad \Gamma_1, \Delta_1 \vdash e : T \quad \Delta_2 = \Delta_1 \cup \{(a, \overline{\Gamma}(T))\} \end{array}}{\Gamma_1, \Delta_1, mss\ a\ \texttt{"="} e \vdash \Gamma_2, \Delta_2}$$

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

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

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 Δ_1 under the name c_i. In Elm these constructors are the only constants beginning with an upper-case letter.

MAYBE-MAIN-SIGN

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

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

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

PROGRAM

Judgment: $prog : T$

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