Typing Rules for Patina

Peter Boyland, Junrui Liu

November 2, 2021

Notation:

- Γ is the *type environment* that maps variables to their types.
- Δ stores the types of all global functions. It maps names of the functions to their argument and return types.
- Judgments of the form "$\Delta; \Gamma \vdash e : \mathsf{T} \dashv \Delta'; \Gamma'$" are *expression typing judgments*. It asserts that, under the environment pair $\Delta; \Gamma$, expression e has type T, and produces a new environment pair $\Delta'; \Gamma'$. Note that in Patina, Δ is not modified under expression typing, so it is always the case that $\Delta = \Delta'$.
  - When the environment Γ is also not modified, we use the abbreviation
  $$\Delta; \Gamma \vdash e : \mathsf{T}$$
  in place of
  $$\Delta; \Gamma \vdash e : \mathsf{T} \dashv \Delta; \Gamma.$$
- Judgments of the form "$\Delta \vdash_{\mathsf{fn}} \mathsf{fn}\, f(x : \mathsf{T}) \to \mathsf{T}_r\ e$" are *function typing judgments*. It asserts that the function f, which takes in an argument x of type T and has body expression e, returns T_r under the global function environment Δ. We only present the typing rules for single-argument functions, but they could be easily extended to account for multi-argument functions.
- Judgments of the form "$\vdash_{\mathsf{prog}} \mathsf{fn_1}, \ldots, \mathsf{fn_n}$" are *program typing judgments*. It asserts that every function in the program is well-typed.

1 Expression Typing Rules (Basic)

The unit value has type Unit:

$$\frac{}{\Delta; \Gamma \vdash () : \mathsf{Unit}}\ \textsc{T-Unit}$$

Boolean constants have type Bool:

$$\frac{}{\Delta; \Gamma \vdash \mathsf{true} : \mathsf{Bool}}\ \textsc{T-True}$$

$$\frac{}{\Delta; \Gamma \vdash \mathsf{false} : \mathsf{Bool}}\ \textsc{T-False}$$

Integer constants have type Int:

$$\frac{i \in \mathbb{Z}}{\Delta; \Gamma \vdash i : \mathsf{Int}}\ \textsc{T-Int}$$

The negation of a boolean expression has type Bool:

$$\frac{\Delta; \Gamma \vdash e : \mathsf{Bool}}{\Delta; \Gamma \vdash !e : \mathsf{Bool}}\ \textsc{T-Not}$$

Binary arithmetic expressions have type Int:

$$\frac{\Delta;\Gamma \vdash e_1 \;:\; \mathsf{Int} \qquad \Delta;\Gamma \vdash e_2 \;:\; \mathsf{Int} \qquad \Box \in \{\mathtt{+},\mathtt{-},\mathtt{*},\mathtt{/}\}}{\Delta;\Gamma \vdash e_1 \Box e_2 \;:\; \mathsf{Int}}\ \text{T-ARITH}$$

Binary logical expressions have type Bool:

$$\frac{\Delta;\Gamma \vdash e_1 \;:\; \mathsf{Bool} \qquad \Delta;\Gamma \vdash e_2 \;:\; \mathsf{Bool} \qquad \Box \in \{\mathtt{\&\&},\mathtt{||}\}}{\Delta;\Gamma \vdash e_1 \Box e_2 \;:\; \mathsf{Bool}}\ \text{T-LOGIC}$$

Integer comparisons have type Bool:

$$\frac{\Delta;\Gamma \vdash e_1 \;:\; \mathsf{Int} \qquad \Delta;\Gamma \vdash e_2 \;:\; \mathsf{Int} \qquad \Box \in \{\mathtt{<},\mathtt{>},\mathtt{<=},\mathtt{>=}\}}{\Delta;\Gamma \vdash e_1 \Box e_2 \;:\; \mathsf{Bool}}\ \text{T-COMPARE}$$

Two expressions of the same type can be checked for (in-)equality:

$$\frac{\Delta;\Gamma \vdash e_1 \;:\; \mathsf{T} \qquad \Delta;\Gamma \vdash e_2 \;:\; \mathsf{T} \qquad \Box \in \{\mathtt{==},\mathtt{!=}\}}{\Delta;\Gamma \vdash e_1 \Box e_2 \;:\; \mathsf{Bool}}\ \text{T-EQ}$$

An if expression has type T, if the condition is a boolean expression and the two branches both have type T:

$$\frac{\Delta;\Gamma \vdash e_1 \;:\; \mathsf{Bool} \qquad \Delta;\Gamma \vdash e_2 \;:\; \mathsf{T} \qquad \Delta;\Gamma \vdash e_3 \;:\; \mathsf{T}}{\Delta;\Gamma \vdash \mathsf{if}\ e_1\ \mathsf{then}\ e_2\ \mathsf{else}\ e_3 \;:\; \mathsf{T}}\ \text{T-IF}$$

A while expression has type Unit, if the condition is a boolean expression and the body is a unit expression.

$$\frac{\Delta;\Gamma \vdash e_1 \;:\; \mathsf{Bool} \qquad \Delta;\Gamma \vdash e_2 \;:\; \mathsf{Unit}}{\Delta;\Gamma \vdash \mathsf{while}\ e_1\, e_2 \;:\; \mathsf{Unit}}\ \text{T-WHILE}$$

2 Expression Typing Rules (Advanced)

An variable x has type T under the type environment Γ, if looking up x in Γ yields T:

$$\frac{\Gamma(x) = \mathsf{T}}{\Delta;\Gamma \vdash x \;:\; \mathsf{T}}\ \text{T-VAR}$$

A let-expression assigns an expression e of type T to the variable x. The let-expression itself has type Unit, and it augments the type environment by mapping x to T:

$$\frac{\Delta;\Gamma \vdash e \;:\; \mathsf{T} \dashv \Delta;\Gamma'}{\Delta;\Gamma \vdash \mathsf{let}\ x : \mathsf{T} = e \;:\; \mathsf{Unit} \dashv \Delta;\Gamma[x \mapsto \mathsf{T}]}\ \text{T-LET}$$

To type-check the sequence $e_1; e_2$, we check that the first expression has type Unit and the second expression has some type T. The type of the overall sequence is T.

$$\frac{\Delta;\Gamma_0 \vdash e_1 \;:\; \mathsf{Unit} \dashv \Delta;\Gamma_1 \qquad \Delta;\Gamma_1 \vdash e_2 \;:\; \mathsf{T} \dashv \Delta;\Gamma_2}{\Delta;\Gamma_0 \vdash e_1; e_2 \;:\; \mathsf{T} \dashv \Delta;\Gamma_2}\ \text{T-SEQ}$$

Note that this rule can be used to type-check arbitrarily long sequences. For example, to type check $e_1; e_2; e_3$ (which should be read as "$e_1; (e_2; e_3)$"), we first check e_1, and then $(e_2; e_3)$.

Remember that a sequence also delineates a scope. To type-check the scope, we check the expressions inside it, but in the end we restore the original environment.

$$\frac{\Delta; \Gamma \vdash e \ : \ \mathsf{T} \dashv \Delta; \Gamma'}{\Delta; \Gamma \vdash \{e\} \ : \ \mathsf{T} \dashv \Delta; \Gamma} \ \textsc{T-Scope}$$

If an variable x has type T in the current scope, then we can assign an expression of the same type to x; the assignment itself has type Unit:

$$\frac{\Gamma(x) = \mathsf{T} \qquad \Delta; \Gamma \vdash e \ : \ \mathsf{T}}{\Delta; \Gamma \vdash x = e \ : \ \mathsf{Unit}} \ \textsc{T-Assign}$$

An integer array can be indexed with an integer index:

$$\frac{\Gamma(x) = \mathsf{Arr} \qquad \Delta; \Gamma \vdash e \ : \ \mathsf{Int}}{\Delta; \Gamma \vdash x[e] \ : \ \mathsf{Int}} \ \textsc{T-Read}$$

An element of an integer array can be overwritten with a new value:

$$\frac{\Gamma(x) = \mathsf{Arr} \qquad \Delta; \Gamma \vdash e_1 \ : \ \mathsf{Int} \qquad \Delta; \Gamma \vdash e_2 \ : \ \mathsf{Int}}{\Delta; \Gamma \vdash x[e_1] = e_2 \ : \ \mathsf{Unit}} \ \textsc{T-Write}$$

The result of calling a function of type $\mathsf{T} \to \mathsf{T}_r$ with an argument of type T has type T_r:

$$\frac{\Delta(f) = \mathsf{T} \to \mathsf{T}_r \qquad \Delta; \Gamma \vdash e \ : \ \mathsf{T}}{\Delta; \Gamma \vdash f(e) \ : \ \mathsf{T}_r} \ \textsc{T-Call}$$

3 Function and Program Typing Rules

To type-check a function definition against Δ (containing types of all global function), we type-check the body of the function, where the initial type environment is just the parameters mapped to their declared types:

$$\frac{\Delta; x : \mathsf{T} \vdash e \ : \ T_r}{\Delta \vdash_{\mathsf{fn}} \mathsf{fn}\, f(x : \mathsf{T}) \to \mathsf{T}_r \ e} \ \textsc{T-Fn}$$

To type-check a program, which is a list of function definitions, we populate Δ with types of all global functions[1], and then type-check each function against Δ:

$$\frac{\begin{array}{c} \mathsf{fn_1} = \mathsf{fn}\, f_1(x : \mathsf{T}_1) \to \mathsf{T}_{r_1} \ e_1 \qquad \cdots \qquad \mathsf{fn_n} = \mathsf{fn}\, f_n(x : \mathsf{T}_n) \to \mathsf{T}_{r_n} \ e_n \\ \Delta \vdash_{\mathsf{fn}} \mathsf{fn}\, f_1(x : \mathsf{T}_1) \to \mathsf{T}_{r_1} \ e_1 \qquad \cdots \qquad \Delta \vdash_{\mathsf{fn}} \mathsf{fn}\, f_n(x : \mathsf{T}_n) \to \mathsf{T}_{r_n} \ e_n \\ \Delta = f_1 : \mathsf{T}_1 \to \mathsf{T}_{r_1}; \ldots ; f_n : \mathsf{T}_n \to \mathsf{T}_{r_n} \end{array}}{\vdash_{\mathsf{prog}} \mathsf{fn_1} \ \ldots \ \mathsf{fn_n}} \ \textsc{T-Prog}$$

[1] This enables us to type-check recursive functions.