The X^3 Language Specification

Ross Tate

October 4, 2013

1 Lexing and Parsing

1.1 Core and Full Languages

ν_v ::= variable/function/method names $\quad$ ν_c ::= class/interface names $\quad$ ν_p ::= type-parameter names $\quad$ $\nu_{vc} ::= \nu_v \mid \nu_c$

kind context	Θ	::=	$\nu_p, \ldots, \nu_p$
type context	Γ	::=	$\nu_v : \tau, \ldots, \nu_v : \tau$
type	τ	::=	$\nu_p \mid \nu_c\langle\tau, \ldots, \tau\rangle \mid \tau \cap \tau \mid \top \mid \bot$
type scheme	σ	::=	$\langle\Theta\rangle(\Gamma) : \tau$
expression	e	::=	$\nu_v \mid \nu_{vc}\langle\tau, \ldots, \tau\rangle(e, \ldots, e) \mid e.\nu_v\langle\tau, \ldots, \tau\rangle(e, \ldots, e) \mid [e, \ldots, e] \mid e +\!\!+ e \mid$ **true** $\mid$ **false** $\mid n \mid$ `"string"`
statement	s	::=	$\{s \ldots s\} \mid \nu_v \coloneqq e; \mid$ **if** (e) s **else** $s \mid$ **while** (e) $s \mid$ **for** $(\nu_v$ **in** $e)$ $s \mid$ **return** e;
interface	i	::=	**interface** $\nu_c\langle\Theta\rangle$ **extends** τ $\{$**fun** $\nu_v\sigma; \ldots;$ **fun** $\nu_v\sigma;\}$
class	c	::=	**class** $\nu_c\langle\Theta\rangle(\Gamma)$ **extends** τ $\{s \ldots s$ **super**$(e, \ldots, e);$ **fun** $\nu_v\sigma$ s $\ldots$ **fun** $\nu_v\sigma$ $s\}$
program	p	::=	$s \mid s \ldots s\ p \mid$ **fun** $\nu_v\sigma$ $s \ldots$ **fun** $\nu_v\sigma$ $s\ p \mid i\ p \mid c\ p$
function context	Δ	::=	$\varnothing \mid \Delta, \nu_{vc}\sigma$
class context	Ψ	::=	$\varnothing \mid \Psi,$ **interface** $\nu_c\langle\Theta\rangle$ **extends** τ $\{\nu_v\sigma; \ldots; \nu_v\sigma;\} \mid \Psi,$ **class** $\nu_c\langle\Theta\rangle$ **extends** τ $\{\nu_v\sigma; \ldots; \nu_v\sigma;\}$

Figure 1: Cubex Core Language Grammar, which gives the grammar for the Cubex core language, along with definitions for the formal constructs Δ and Ψ, which are not part of the language per se but are useful in our formalization of the type system. Note that p stands for "program" and defines a syntactically-valid program in the Cubex core language. Note also that lists, represented with an elipsis, may consist of zero, one, or more elements. So $a, \ldots, a$ may be the empty string, a, or some list of as separated by commas.

We distinguish between the Cubex *core* language, which is specified by the grammar in Figure 1.1, and the *full* language. The full language differs from the core language in a number of ways:

- It includes the following unary and binary operators, listed in order of precedence, which are (except for `++`) short for method calls on arbitrary classes (their signatures are those that are given for the built-in classes at the end of this document). All operators are left-associative.
  1. Unary prefixes `-` and `!` short for `negative` and `negate` respectively.
  2. Multiplicative operators `*`, `/`, and `%` short for `times`, `divide`, and `modulo` respectively.
  3. Additive `+` and `-` short for `plus` and `minus` respectively.
  4. Range operators `..`, `<.`, `.<`, `<<`, `...`, and `<..`, with the last two being unary suffixes; the binary operators are shorthand for `through` using additional `Boolean` parameters to indicate whether to include the lower and/or upper bounds; the unary suffixes are shorthand for `onwards` using an additional `Boolean` parameter to indicate inclusiveness.
  5. The binary operator `++`, (actually part of the core language), which implements appending of iterables, denoted in the grammar by $+\!\!+$.
  6. Inequality operators `<`, `<=`, `>=`, and `>` short for `lessThan` with an additional `Boolean` parameter indicating strictness, and with order reversed for `>=` and `>`.
  7. Equational operators `==` and `!=`, short for `equals` and `equals` followed by `negate`.
  8. Boolean operators `&` short for `and` followed by `|` short for `or`.
- Characters in the core language which have no obvious ASCII equivalent are represented in the full language in ASCII as follows:

  - `Thing` for $\top$ and `Nothing` for $\bot$.
  - `&` for $\wedge$ and `|` for $\vee$.
  - `&` for $\cap$.
  - `<` for $\langle$ and `>` for $\rangle$.

- Expressions (denoted e in the grammar) may be surrounded by parentheses.
- If there are no type parameters (i.e. when $\langle \tau, \ldots, \tau \rangle$ is the empty list), a class/interface/function specification can drop the $\langle\rangle$.
- When calling a function/method/constructor with no type parameters, the $\langle\rangle$ may be dropped.
- If the **else** case is {}, it can be dropped.
- More importantly, interfaces can provide method implementations, and classes can omit method implementations if some inherited class or interface provides an implementation (taking the implementation for the first such class or interface in the extended intersection type).
- If a class or interface just extends $\top$, the **extends** clause can be omitted.
- If the call to **super** has no arguments, it can be omitted.
- If a function implementation is simply **return** of some expression, the **return** can be abbreviated to =. For example,

  ```
  fun foo(y : Integer) : Integer return y * 2;
  ```

  may be abbreviated to

  ```
  fun foo(y : Integer) : Integer = y * 2;
  ```

Finally, there is no name hiding. If a judgement requires binding a name twice, that judgement is ill formed and does not hold.

1.2 Name, Keywords, Literals, etc.

The grammar uses ν with subscripts to denote both various kinds of names. We enforce a distinction between these kinds. Variables and function/method names (ν_v) are a lower-case letter followed by letters, numbers, and/or underscores. Class/interface names (ν_c) are an upper-case letter followed by one or more letters, numbers, and/or underscores. Type parameters (ν_p) are an upper-case letter alone. In the formalism we refer to all of these collectively as just ν.

Key words (i.e. any words receiving special treatment in the core language) cannot be used as names or variables.

String literals (denoted in the grammar by `"string"`) have no escaping and cannot contain white space besides the space character. Integer literals (denoted in the grammar by n) are in decimal only.

Finally, note that the grammar contains various other syntactic artifacts (semicolons, brackets, etc.) whose only legal uses in the language (outside of comments and string literals) are fully specified by their role in the grammar.

1.3 Whitespace and Comments

Unless it occurs inside a string literal, white space is disregarded except for its role of separating names and other tokens (e.g. "`: =`" is not the same as "`:=`" and only the latter can be used in a variable assignment). Unless it occurs inside a string literal, `#` starts a comment that extends to the end of the line. Unless it occurs inside a string literal, `‘` starts a comment that extends to the matching `’`, where these comments can be nested inside each other. Comments are disregarded except for their role of separating names and other tokens.

2 Validating

The following formalizes when a program is valid, incorporating type validity, type checking, and class/interface validation. Judgements take the form "(context) $\vdash$ (property)". Ψ indicates the classes and interfaces in scope, what they directly inherit, and what their methods are. Θ indicates the type parameters in scope. Δ indicates the function names in scope and what their type scheme is. Γ typically indicates the immutable variable names in scope and what their type is. $\hat{\Gamma}$ typically indicates the mutable variable names in scope and what their type is. The following is a summary of the judgements used in this formalism and where their definitions can be found.

Judgement	Meaning	Figure
$\Psi \mid \Theta \vdash \tau <: \tau'$	τ is a subtype of τ'	2
$\Psi \vdash \nu\langle\Theta\rangle$ **extends** τ	generic class/interface $\nu\langle\Theta\rangle$ directly inherits τ	2
$\Psi \mid \Theta \vdash \tau.\nu : \sigma$	method ν of type τ has type scheme σ	3
$\Psi \vdash \hat{\nu}\langle\Theta\rangle.\nu : \sigma$	generic class/interface $\hat{\nu}\langle\Theta\rangle$ has method ν with type scheme σ	3
$\Psi \mid \Theta \vdash \tau$	τ is a valid type	4
$\Psi \mid \Theta \vdash_{\hat{\tau}} \tau$	τ is an inheritable type with constructable component $\hat{\tau}$	4
$\Psi \mid \Theta \vdash \Gamma$	Γ is a valid type context	4
$\Psi \mid \Theta \vdash \sigma$	σ is a valid type scheme	4
$\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e : \tau$	expression e has type τ	5
$\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash s : \hat{\Gamma}'$	mutable variables $\hat{\Gamma}'$ are available after valid statement s	6
$\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^b_\tau s : \hat{\Gamma}'$	like above and all returns have type τ with $b =$ **true** guaranteeing a return	7
$\Psi \mid \Delta \mid \Gamma \vdash i : \Psi'$	i is a valid interface with signature Ψ'	8
$\Psi \mid \Delta \mid \Gamma \vdash c : \Psi' \mid \Delta'$	c is a valid class with signature Ψ' and constructor Δ'	8
$\Psi \mid \Delta \mid \Gamma \vdash p$	p is a valid program	9
$\vdash p$	p is a valid program in the initial environment	9

$\boxed{\Psi \mid \Theta \vdash \tau <: \tau \qquad \Psi \vdash \nu\langle\Theta\rangle \textbf{ extends } \tau}$

$$\frac{}{\Psi \mid \Theta \vdash \nu <: \nu} \qquad \frac{}{\Psi \mid \Theta \vdash \bot <: \tau} \qquad \frac{}{\Psi \mid \Theta \vdash \tau <: \top}$$

$$\frac{\Psi \mid \Theta \vdash \tau_i <: \tau}{\Psi \mid \Theta \vdash \tau_1 \cap \tau_2 <: \tau} \qquad \frac{\Psi \mid \Theta \vdash \tau <: \tau_1 \qquad \Psi \mid \Theta \vdash \tau <: \tau_2}{\Psi \mid \Theta \vdash \tau <: \tau_1 \cap \tau_2}$$

$$\frac{\text{for all } i, \quad \Psi \mid \Theta \vdash \tau_i <: \tau_i' \quad \text{and} \quad \Psi \mid \Theta \vdash \tau_i' <: \tau_i}{\Psi \mid \Theta \vdash \nu\langle\tau_1, \ldots, \tau_n\rangle <: \nu\langle\tau_1', \ldots, \tau_n'\rangle} \qquad \frac{\Psi \mid \Theta \vdash \tau <: \tau'}{\Psi \mid \Theta \vdash \texttt{Iterable}\langle\tau\rangle <: \texttt{Iterable}\langle\tau'\rangle}$$

$$\frac{\Psi \vdash \nu\langle\nu_1, \ldots, \nu_n\rangle \textbf{ extends } \tau' \qquad \Psi \mid \Theta \vdash \tau'[\nu_1 \mapsto \tau_1, \ldots, \nu_n \mapsto \tau_n] <: \tau}{\Psi \mid \Theta \vdash \nu\langle\tau_1, \ldots \tau_n\rangle <: \tau} \qquad \frac{\textbf{interface } \nu\langle\Theta\rangle \textbf{ extends } \tau \ \{\ldots\} \text{ in } \Psi}{\Psi \vdash \nu\langle\Theta\rangle \textbf{ extends } \tau} \qquad \frac{\textbf{class } \nu\langle\Theta\rangle \textbf{ extends } \tau \ \{\ldots\} \text{ in } \Psi}{\Psi \vdash \nu\langle\Theta\rangle \textbf{ extends } \tau}$$

Figure 2: Subtyping

$\boxed{\Psi \mid \Theta \vdash \tau.\nu : \sigma \qquad \Psi \vdash \nu\langle\Theta\rangle.\nu : \sigma}$

$$\frac{\Psi \mid \Theta \vdash \tau'.\nu : \sigma \qquad \Psi \mid \Theta \vdash \tau <: \tau'}{\Psi \mid \Theta \vdash \tau.\nu : \sigma} \qquad \frac{}{\Psi \mid \Theta \vdash \bot.\nu : \sigma}$$

$$\frac{\Psi \vdash \hat{\nu}\langle\nu_1, \ldots, \nu_n\rangle.\nu : \sigma}{\Psi \mid \Theta \vdash \hat{\nu}\langle\tau_1, \ldots, \tau_n\rangle.\nu : \sigma[\nu_1 \mapsto \tau_1, \ldots, \nu_n \mapsto \tau_n]}$$

$$\frac{\textbf{interface } \hat{\nu}\langle\Theta\rangle \textbf{ extends } \tau \ \{\ldots \nu\sigma; \ldots\} \text{ in } \Psi}{\Psi \vdash \hat{\nu}\langle\Theta\rangle.\nu : \sigma} \qquad \frac{\textbf{class } \hat{\nu}\langle\Theta\rangle \textbf{ extends } \tau \ \{\ldots \nu\sigma; \ldots\} \text{ in } \Psi}{\Psi \vdash \hat{\nu}\langle\Theta\rangle.\nu : \sigma}$$

Figure 3: Method Lookup

$\boxed{\Psi \mid \Theta \vdash \tau \qquad \Psi \mid \Theta \vdash_\tau \tau \qquad \Psi \mid \Theta \vdash \Gamma \qquad \Psi \mid \Theta \vdash \sigma}$

$$\frac{\Psi \mid \Theta \vdash_{\hat{\tau}} \tau}{\Psi \mid \Theta \vdash \tau} \qquad \frac{}{\Psi \mid \Theta \vdash_\top \top} \qquad \frac{}{\Psi \mid \Theta \vdash \bot} \qquad \frac{\nu \text{ in } \Theta}{\Psi \mid \Theta \vdash \nu}$$

$$\frac{\textbf{interface } \nu\langle \nu_1, \ldots, \nu_n \rangle \textbf{ extends } \tau \ \{\ldots\} \text{ in } \Psi \quad \text{for all } i, \quad \Psi \mid \Theta \vdash \tau_i}{\Psi \mid \Theta \vdash_\top \nu\langle \tau_1, \ldots, \tau_n \rangle}$$

$$\frac{\textbf{class } \nu\langle \nu_1, \ldots, \nu_n \rangle \textbf{ extends } \tau \ \{\ldots\} \text{ in } \Psi \quad \text{for all } i, \quad \Psi \mid \Theta \vdash \tau_i}{\Psi \mid \Theta \vdash_{\nu\langle \tau_1, \ldots, \tau_n \rangle} \nu\langle \tau_1, \ldots, \tau_n \rangle}$$

$$\frac{\begin{array}{lll} & \Psi \mid \Theta \vdash_{\hat{\tau}} \tau & \Psi \mid \Theta \vdash_\top \tau' \\ \text{for all } \nu, & \left(\begin{array}{c} \Psi \mid \Theta \vdash \tau <: \nu\langle \tau_1, \ldots, \tau_n \rangle \\ \text{and} \\ \Psi \mid \Theta \vdash \tau' <: \nu\langle \tau'_1, \ldots, \tau'_n \rangle \end{array}\right) & \text{implies} \quad \left(\begin{array}{c} \Psi \mid \Theta \vdash \tau <: \nu\langle \tau'_1, \ldots, \tau'_n \rangle \\ \text{and} \\ \Psi \mid \Theta \vdash \tau' <: \nu\langle \tau_1, \ldots, \tau_n \rangle \end{array}\right) \\ \text{for all } \nu, & \Psi \mid \Theta \vdash \tau.\nu : \sigma \quad \text{and} & \Psi \mid \Theta \vdash \tau'.\nu : \sigma' \quad \text{implies} \quad \sigma = \sigma' \end{array}}{\Psi \mid \Theta \vdash_{\hat{\tau}} \tau \cap \tau'}$$

$$\frac{\text{for all } i, \quad \Psi \mid \Theta \vdash \tau_i}{\Psi \mid \Theta \vdash \nu_1 : \tau_1, \ldots, \nu_n : \tau_n} \qquad \frac{\Psi \mid \Theta, \hat{\Theta} \vdash \Gamma \quad \Psi \mid \Theta, \hat{\Theta} \vdash \tau}{\Psi \mid \Theta \vdash \langle \hat{\Theta} \rangle(\Gamma) : \tau}$$

Figure 4: Type Validity

$\boxed{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e : \tau}$

$$\frac{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e : \tau \quad \Psi \mid \Theta \vdash \tau <: \tau'}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e : \tau'}$$

$$\frac{\nu : \tau \text{ in } \Gamma}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash \nu : \tau}$$

$$\frac{\begin{array}{c} \nu\langle \nu_1, \ldots, \nu_m \rangle(\hat{\nu}_1 : \hat{\tau}_1, \ldots, \hat{\nu}_n : \hat{\tau}_n) : \tau \text{ in } \Delta \\ \text{for all } i, \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e_i : \hat{\tau}_i[\nu_1 \mapsto \tau_1, \ldots, \nu_m \mapsto \tau_m] \end{array}}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash \nu\langle \tau_1, \ldots, \tau_m \rangle(e_1, \ldots, e_n) : \tau[\nu_1 \mapsto \tau_1, \ldots, \nu_m \mapsto \tau_m]}$$

$$\frac{\begin{array}{c} \Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e : \hat{\tau} \\ \Psi \mid \Theta \vdash \hat{\tau}.\nu : \langle \nu_1, \ldots, \nu_m \rangle(\hat{\nu}_1 : \hat{\tau}_1, \ldots, \hat{\nu}_n : \hat{\tau}_n) : \tau \\ \text{for all } i, \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e_i : \hat{\tau}_i[\nu_1 \mapsto \tau_1, \ldots, \nu_m \mapsto \tau_m] \end{array}}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e.\nu\langle \tau_1, \ldots, \tau_m \rangle(e_1, \ldots, e_n) : \tau[\nu_1 \mapsto \tau_1, \ldots, \nu_m \mapsto \tau_m]}$$

$$\frac{\text{for all } i, \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e_i : \tau}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash [e_1, \ldots, e_n] : \texttt{Iterable}\langle \tau \rangle} \qquad \frac{\text{for all } i, \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e_i : \texttt{Iterable}\langle \tau \rangle}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash e_1 +\!\!+ e_2 : \texttt{Iterable}\langle \tau \rangle}$$

$$\frac{}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash \textbf{true} : \texttt{Boolean}\langle\rangle} \qquad \frac{}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash \textbf{false} : \texttt{Boolean}\langle\rangle}$$

$$\frac{}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash n : \texttt{Integer}\langle\rangle} \qquad \frac{}{\Psi \mid \Theta \mid \Delta \mid \Gamma \vdash \texttt{"string"} : \texttt{String}\langle\rangle}$$

Figure 5: Type Checking Expressions

$\boxed{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \Gamma \vdash s : \Gamma}$

$$\frac{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash s : \hat{\Gamma}', \nu : \tau, \hat{\Gamma}''}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash s : \hat{\Gamma}', \hat{\Gamma}''} \qquad \frac{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash s : \hat{\Gamma}', \nu : \tau, \hat{\Gamma}'' \quad \Psi \mid \Theta \vdash \tau <: \tau'}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash s : \hat{\Gamma}', \nu : \tau', \hat{\Gamma}''}$$

$$\frac{\text{for all } i, \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma}_{i-1} \vdash s_i : \hat{\Gamma}_i}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma}_0 \vdash \{s_1 \dots s_n\} : \hat{\Gamma}_n}$$

$$\frac{\Psi \mid \Theta \mid \Delta \mid \Gamma, \hat{\Gamma} \vdash e : \tau}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash \nu \coloneqq e; : \hat{\Gamma}, \nu : \tau} \qquad \frac{\Psi \mid \Theta \mid \Delta \mid \Gamma, \hat{\Gamma}, \nu : \tau, \hat{\Gamma}' \vdash e : \tau'}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma}, \nu : \tau, \hat{\Gamma}' \vdash \nu \coloneqq e; : \hat{\Gamma}, \nu : \tau', \hat{\Gamma}'}$$

$$\frac{\Psi \mid \Theta \mid \Delta \mid \Gamma, \hat{\Gamma} \vdash e : \texttt{Boolean}\langle\rangle \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash s_1 : \hat{\Gamma}' \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash s_2 : \hat{\Gamma}'}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash \textbf{if } (e) \ s_1 \textbf{ else } s_2 : \hat{\Gamma}'}$$

$$\frac{\Psi \mid \Theta \mid \Delta \mid \Gamma, \hat{\Gamma} \vdash e : \texttt{Boolean}\langle\rangle \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash s : \hat{\Gamma}}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash \textbf{while } (e) \ s : \hat{\Gamma}}$$

$$\frac{\Psi \mid \Theta \mid \Delta \mid \Gamma, \hat{\Gamma} \vdash e : \texttt{Iterable}\langle\tau\rangle \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma}, \nu : \tau \vdash s : \hat{\Gamma}}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash \textbf{for } (\nu \textbf{ in } e) \ s : \hat{\Gamma}}$$

Figure 6: Type Checking Statements

$\boxed{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \Gamma \vdash^{b}_{\tau} s : \Gamma}$

$$\frac{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^{\textbf{true}}_{\tau} s : \hat{\Gamma}'}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^{\textbf{false}}_{\tau} s : \hat{\Gamma}'}$$

$$\frac{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^{b}_{\tau} s : \hat{\Gamma}', \nu : \hat{\tau}, \hat{\Gamma}''}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^{b}_{\tau} s : \hat{\Gamma}', \hat{\Gamma}''} \qquad \frac{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^{b}_{\tau} s : \hat{\Gamma}', \nu : \hat{\tau}, \hat{\Gamma}'' \quad \Psi \mid \Theta \vdash \hat{\tau} <: \hat{\tau}'}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^{b}_{\tau} s : \hat{\Gamma}', \nu : \hat{\tau}', \hat{\Gamma}''}$$

$$\frac{\text{for all } i, \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma}_{i-1} \vdash^{b_i}_{\tau} s_i : \hat{\Gamma}_i \quad b_n \text{ implies } n > 0}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma}_0 \vdash^{b_n}_{\tau} \{s_1 \dots s_n\} : \hat{\Gamma}_n}$$

$$\frac{\Psi \mid \Theta \mid \Delta \mid \Gamma, \hat{\Gamma} \vdash e : \hat{\tau}}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^{\textbf{false}}_{\tau} \nu \coloneqq e; : \hat{\Gamma}, \nu : \hat{\tau}} \qquad \frac{\Psi \mid \Theta \mid \Delta \mid \Gamma, \hat{\Gamma}, \nu : \hat{\tau}, \hat{\Gamma}' \vdash e : \hat{\tau}'}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma}, \nu : \hat{\tau}, \hat{\Gamma}' \vdash^{\textbf{false}}_{\tau} \nu \coloneqq e; : \hat{\Gamma}, \nu : \hat{\tau}', \hat{\Gamma}'}$$

$$\frac{\Psi \mid \Theta \mid \Delta \mid \Gamma, \hat{\Gamma} \vdash e : \texttt{Boolean}\langle\rangle \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^{b}_{\tau} s_1 : \hat{\Gamma}' \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^{b}_{\tau} s_2 : \hat{\Gamma}'}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^{b}_{\tau} \textbf{if } (e) \ s_1 \textbf{ else } s_2 : \hat{\Gamma}'}$$

$$\frac{\Psi \mid \Theta \mid \Delta \mid \Gamma, \hat{\Gamma} \vdash e : \texttt{Boolean}\langle\rangle \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^{b}_{\tau} s : \hat{\Gamma}}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^{\textbf{false}}_{\tau} \textbf{while } (e) \ s : \hat{\Gamma}}$$

$$\frac{\Psi \mid \Theta \mid \Delta \mid \Gamma, \hat{\Gamma} \vdash e : \texttt{Iterable}\langle\hat{\tau}\rangle \quad \Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma}, \nu : \hat{\tau} \vdash^{b}_{\tau} s : \hat{\Gamma}}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^{\textbf{false}}_{\tau} \textbf{for } (\nu \textbf{ in } e) \ s : \hat{\Gamma}}$$

$$\frac{\Psi \mid \Theta \mid \Delta \mid \Gamma, \hat{\Gamma} \vdash e : \tau}{\Psi \mid \Theta \mid \Delta \mid \Gamma \mid \hat{\Gamma} \vdash^{\textbf{true}}_{\tau} \textbf{return } e; : \hat{\Gamma}}$$

Figure 7: Type Checking Returns

$$\boxed{\Psi \mid \Delta \mid \Gamma \vdash i : \Psi \qquad \Psi \mid \Delta \mid \Gamma \vdash c : \Psi \mid \Delta}$$

$$\frac{\begin{array}{c}\Psi \mid \Theta \vdash_{\top} \tau \\ \Psi' = \textbf{interface}\ \nu\langle\Theta\rangle\ \textbf{extends}\ \tau\ \{\nu_1\sigma_1; \ldots; \nu_n\sigma_n;\} \\ \text{for all } i, \qquad \Psi, \Psi' \mid \Theta \vdash \sigma_i \\ \text{for all } i, \qquad \Psi \mid \Theta \vdash \tau.\nu_i : \sigma \qquad \text{implies} \qquad \sigma = \sigma_i\end{array}}{\Psi \mid \Delta \mid \Gamma \vdash \textbf{interface}\ \nu\langle\Theta\rangle\ \textbf{extends}\ \tau\ \{\textbf{fun}\ \nu_1\sigma_1; \ldots; \textbf{fun}\ \nu_n\sigma_n;\} : \Psi'}$$

$$\frac{\begin{array}{c}\Psi \mid \Theta \vdash_{\hat{\tau}} \tau \\ \Psi' = \textbf{class}\ \nu\langle\Theta\rangle\ \textbf{extends}\ \tau\ \{\nu_1\langle\Theta_1\rangle(\Gamma_1); \ldots; \nu_n\langle\Theta_n\rangle(\Gamma_n);\} \qquad \Delta' = \nu\langle\Theta\rangle(\hat{\Gamma}) : \nu\langle\Theta\rangle \\ \Psi, \Psi' \mid \Theta \vdash \hat{\Gamma} \qquad \hat{\Gamma}_0 = \hat{\Gamma} \qquad \text{for all } i, \qquad \Psi, \Psi' \mid \Theta \mid \Delta, \Delta' \mid \Gamma \mid \hat{\Gamma}_{i-1} \vdash s_i : \hat{\Gamma}_i \\ \tau(e_1, \ldots, e_k) = \top() \qquad \text{or} \qquad \Psi, \Psi' \mid \Theta \mid \Delta, \Delta' \mid \Gamma, \hat{\Gamma}_n \vdash \hat{\tau}(e_1, \ldots, e_k) : \hat{\tau} \\ \Delta'' = \Delta, \Delta', \nu_1\langle\Theta_1\rangle(\Gamma_1), \ldots, \nu_n\langle\Theta_n\rangle(\Gamma_n) \\ \text{for all } i, \quad \Psi, \Psi' \mid \Theta \vdash \sigma_i \quad \sigma_i = \langle\Theta_i\rangle(\Gamma_i) : \tau_i \quad \Psi, \Psi' \mid \Theta, \Theta_i \mid \Delta'' \mid \Gamma, \hat{\Gamma}_n \mid \Gamma_i \vdash^{\textbf{true}}_{\tau_i} \hat{s}_i : \Gamma'_i \\ \text{for all } \hat{\nu}\ \sigma, \qquad \Psi \mid \Theta \vdash \tau.\hat{\nu} : \sigma \qquad \text{implies} \qquad \text{exists } i, \qquad \nu_i = \hat{\nu} \qquad \sigma_i = \sigma\end{array}}{\Psi \mid \Delta \mid \Gamma \vdash \textbf{class}\ \nu\langle\Theta\rangle(\hat{\Gamma})\ \textbf{extends}\ \tau\ \{s_1 \ldots s_m\ \textbf{super}(e_1, \ldots, e_k);\ \textbf{fun}\ \nu_1\sigma_1\ \hat{s}_1\ \ldots\ \textbf{fun}\ \nu_n\sigma_n\ \hat{s}_n\} : \Psi' \mid \Delta'}$$

Figure 8: Class and Interface Checking

$\boxed{\Psi \mid \Delta \mid \Gamma \vdash p \qquad \vdash p}$

$$\frac{\Psi \mid \varnothing \mid \Delta \mid \Gamma \mid \varnothing \vdash^{\mathbf{true}}_{\texttt{Iterable}\langle\texttt{String}\rangle} s : \hat{\Gamma}}{\Psi \mid \Delta \mid \Gamma \vdash s}$$

$$\frac{\hat{\Gamma}_0 = \varnothing \quad \text{for all } i, \quad \Psi \mid \varnothing \mid \Delta \mid \Gamma \mid \hat{\Gamma}_{i-1} \vdash^{b_i}_{\texttt{Iterable}\langle\texttt{String}\rangle} s_i : \hat{\Gamma}_i \quad \Psi \mid \Delta \mid \Gamma, \hat{\Gamma}_n \vdash p}{\Psi \mid \Delta \mid \Gamma \vdash s_1 \ldots s_n \; p}$$

$$\frac{\begin{array}{c}\Delta' = \Delta, \nu_1\langle\Theta_1\rangle(\Gamma_1) : \tau_1, \ldots, \nu_n\langle\Theta_n\rangle(\Gamma_n) : \tau_n \\ \text{for all } i, \quad \Psi \mid \Theta_i \vdash \Gamma_i \quad \Psi \mid \Theta_i \vdash \tau_i \quad \Psi \mid \Theta_i \mid \Delta' \mid \Gamma \mid \Gamma_i \vdash^{\mathbf{true}}_{\tau_i} s_i : \hat{\Gamma}_i \\ \Psi \mid \Delta' \mid \Gamma \vdash p\end{array}}{\Psi \mid \Delta \mid \Gamma \vdash \mathbf{fun}\ \nu_1\langle\Theta_1\rangle(\Gamma_1) : \tau_1 \; s_1 \ldots \mathbf{fun}\ \nu_n\langle\Theta_n\rangle(\Gamma_n) : \tau_n \; s_n \; p}$$

$$\frac{\Psi \mid \Delta \mid \Gamma \vdash i : \Psi' \quad \Psi, \Psi' \mid \Delta \mid \Gamma \vdash p}{\Psi \mid \Delta \mid \Gamma \vdash i \; p} \qquad \frac{\Psi \mid \Delta \mid \Gamma \vdash c : \Psi' \mid \Delta' \quad \Psi, \Psi' \mid \Delta, \Delta' \mid \Gamma \vdash p}{\Psi \mid \Delta \mid \Gamma \vdash c \; p}$$

$\Psi_0 =$ **class** Iterable⟨E⟩ **extends** ⊤ { },
class Boolean⟨⟩ **extends** ⊤ {
negate⟨⟩() : Boolean⟨⟩;
and⟨⟩(that : Boolean⟨⟩) : Boolean⟨⟩;
or⟨⟩(that : Boolean⟨⟩) : Boolean⟨⟩;
through⟨⟩(upper : Boolean⟨⟩, includeLower : Boolean⟨⟩, includeUpper : Boolean⟨⟩) : Iterable⟨Boolean⟨⟩⟩;
onwards⟨⟩(inclusive : Boolean⟨⟩) : Iterable⟨Boolean⟨⟩⟩;
lessThan⟨⟩(that : Boolean⟨⟩, strict : Boolean⟨⟩) : Boolean⟨⟩;
equals⟨⟩(that : Boolean⟨⟩) : Boolean⟨⟩;
},
class Integer⟨⟩ **extends** ⊤ {
negative⟨⟩() : Integer⟨⟩;
times⟨⟩(factor : Integer⟨⟩) : Integer⟨⟩;
divide⟨⟩(divisor : Integer⟨⟩) : Iterable⟨Integer⟨⟩⟩;
modulo⟨⟩(modulus : Integer⟨⟩) : Iterable⟨Integer⟨⟩⟩;
plus⟨⟩(summand : Integer⟨⟩) : Integer⟨⟩;
minus⟨⟩(subtrahend : Integer⟨⟩) : Integer⟨⟩;
through⟨⟩(upper : Integer⟨⟩, includeLower : Boolean⟨⟩, includeUpper : Boolean⟨⟩) : Iterable⟨Integer⟨⟩⟩;
onwards⟨⟩(inclusive : Boolean⟨⟩) : Iterable⟨Integer⟨⟩⟩;
lessThan⟨⟩(that : Integer⟨⟩, strict : Boolean⟨⟩) : Boolean⟨⟩;
equals⟨⟩(that : Integer⟨⟩) : Boolean⟨⟩;
},
class Character⟨⟩ **extends** ⊤ {
unicode⟨⟩() : Integer⟨⟩;
equals⟨⟩(that : Character⟨⟩) : Boolean⟨⟩;
},
class String⟨⟩ **extends** Iterable⟨Character⟨⟩⟩ { equals⟨⟩(that : String⟨⟩) : Boolean⟨⟩; }
$\Delta_0 =$ character⟨⟩(unicode : Integer⟨⟩) : Character⟨⟩, string⟨⟩(characters : Iterable⟨Character⟩) : String⟨⟩
$\Gamma_0 =$ input : Iterable⟨String⟨⟩⟩

$$\frac{\Psi_0 \mid \Delta_0 \mid \Gamma_0 \vdash p}{\vdash p}$$

Figure 9: Program Checking