

Oat v. 2 Language Specification

CIS341 – Steve Zdancewic

March 23, 2026

1 Grammar

The following grammar defines the Oat syntax. All binary operations are *left associative* with precedence levels indicated numerically. Higher precedence operators bind tighter than lower precedence ones. Here *id* ranges over lower-case identifiers and *S* indicates an upper-case “struct name.” The parts of the grammar marked with M (for Meta) are not part of the abstract syntax, but are included for parsing purposes (e.g. parentheses) or for use in explaining the typing judgments.

Some of the rules mention the symbols \top (true) and \perp (false), which act like boolean flags that indicate (1) whether a statement or block returns, or (2) whether a left-hand-side is assignable.

<i>t</i>	::=	types
	<code>int</code>	
	<code>bool</code>	
	<code>ref?</code>	nullable references
	<code>ref</code>	non-null references
<i>ref</i>	::=	reference types
	<code>string</code>	
	<i>S</i>	named (mutable) record type
	<code>t []</code>	arrays
	<code>(<i>t</i>₀, ..., <i>t</i>_{<i>n</i>}) -> <i>rt</i></code>	function pointer
	<code>(<i>ref</i>)</code>	
<i>rt</i>	::=	return types
	<code>void</code>	
	<i>t</i>	
<i>prog</i>	::=	prog
	ϵ	
	<code>decl prog</code>	
<i>decl</i>	::=	global declarations
	<code>gdecl</code>	
	<code>fdecl</code>	
	<code>tdecl</code>	

<i>gdecl</i>	::= <code>global id = gexp;</code>	global value declarations
<i>arg</i>	::= <code>t id</code>	arg
<i>args</i>	::= <code>arg₁, .., arg_n</code>	args
<i>fdecl</i>	::= <code>rt id(args) block</code>	function declaration
<i>field</i>	::= <code>t x</code>	field declarations
<i>fields</i>	::= <code>field₁; .. ;field_n</code>	fields
<i>tdecl</i>	::= <code>struct S{ fields }</code>	struct declaration
<i>exp</i>	::= <code>0</code> <code>ref null</code> <code>true</code> <code>false</code> <code>integer</code> <code>string</code> <code>lhs</code> <code>new t[] {exp₁, .., exp_n}</code> <code>new t [exp]</code> <code>new t [exp₁] {id -> exp₂}</code> <code>length (exp)</code> <code>new S {x₁=exp₁; .. ; x_n=exp_n}</code> <code>exp (exp₁, .., exp_n)</code> <code>uop exp</code> <code>exp₁ bop exp₂</code> <code>(exp)</code>	expressions literal 0 type-annotated null value 64-bit integer literals C-style strings left-hand-side as an expression array literal value array with default initializer array with explicit initializer built-in polymorphic length operation struct initialization function call unary operation binary operation M (only for parsing)
<i>lhs</i>	::= <code>id</code> <code>exp . id</code> <code>lhs [exp₂]</code> <code>exp (exp₁, .., exp_n) [exp₂]</code>	left-hand-sides for assignment identifier, resolved to global, local, or function name field projection M (parsed as <code>exp[exp₂]</code>) M (parsed as <code>exp[exp₂]</code>)
<i>gexp</i>	::= <code>integer</code>	global initializers 64-bit integer literals

	<pre> string ref null true false new t[] {gexp₁, .., gexp_n} new S{x₁=gexp₁; .. ;x_n=gexp_n} </pre>	C-style strings
<i>block</i>	<pre> ::= {stmt₁ .. stmt_n} </pre>	blocks
<i>vdecl</i>	<pre> ::= var id = exp </pre>	local variable declarations
<i>stmt</i>	<pre> ::= lhs = exp; vdecl; return exp; return; exp(exp₁, .., exp_n); if_stmt for(vdecls; exp_{opt}; stmt_{opt}) block while(exp) block </pre>	statements assignment statement variable declaration return with value return (void) call a void-returning function conditionals
<i>if_stmt</i>	<pre> ::= if(exp) block else_stmt if?(ref id = exp) block else_stmt </pre>	if statements standard boolean if possibly-null checked downcast
<i>else_stmt</i>	<pre> ::= ε else block else if_stmt </pre>	else

<i>bop</i>	::=	(left associative) binary operations
		* multiplication (precedence 100)
		+ addition (precedence 90)
		- subtraction (precedence 90)
		<< shift left (precedence 80)
		>> shift right logical (precedence 80)
		>>> shift right arithmetic (precedence 80)
		< less-than (precedence 70)
		<= less-than or equal (precedence 70)
		> greater-than (precedence 70)
		>= greater-than or equal (precedence 70)
		== equal (precedence 60)
		!= not equal (precedence 60)
		& logical and (precedence 50)
		logical or (precedence 40)
		[&] bit-wise and (precedence 30)
		[] bit-wise or (precedence 20)

<i>uop</i>	::=	unary operations
		-
		!
		~

2 Subtyping Rules

$$\boxed{H \vdash t_1 \leq t_2}$$

$$\frac{}{H \vdash \text{int} \leq \text{int}} \text{SUB_SUB_INT}$$

$$\frac{}{H \vdash \text{bool} \leq \text{bool}} \text{SUB_SUB_BOOL}$$

$$\frac{H \vdash_r \text{ref}_1 \leq \text{ref}_2}{H \vdash \text{ref}_1? \leq \text{ref}_2?} \text{SUB_SUB_NREF}$$

$$\frac{H \vdash_r \text{ref}_1 \leq \text{ref}_2}{H \vdash \text{ref}_1 \leq \text{ref}_2} \text{SUB_SUB_REF}$$

$$\frac{H \vdash_r \text{ref}_1 \leq \text{ref}_2}{H \vdash \text{ref}_1 \leq \text{ref}_2?} \text{SUB_SUB_NRREF}$$

$$\boxed{H \vdash_r \text{ref}_1 \leq \text{ref}_2}$$

$$\frac{}{H \vdash_r \text{string} \leq \text{string}} \text{SUB_SUBR_STRING}$$

$$\frac{}{H \vdash_r t[] \leq t[]} \text{SUB_SUBR_ARRAY}$$

$$\frac{\text{struct } S_1\{t_1 x_1; \dots; t_n x_n; t_{n+1} x_{n+1}; \dots; t_m x_m\} \in H \quad \text{struct } S_2\{t_1 x_1; \dots; t_n x_n\} \in H}{H \vdash_r S_1 \leq S_2} \text{SUB_SUBR_STRUCT}$$

$$\frac{H \vdash t'_1 \leq t_1 \quad \dots \quad H \vdash t'_n \leq t_n \quad H \vdash_{rt} rt_1 \leq rt_2}{H \vdash_r (t_1, \dots, t_n) \rightarrow rt_1 \leq (t'_1, \dots, t'_n) \rightarrow rt_2} \text{SUB_SUBR_FUNT}$$

$$\boxed{H \vdash_{rt} rt_1 \leq rt_2}$$

$$\frac{}{H \vdash_{rt} \text{void} \leq \text{void}} \text{SUB_SUBRET_SVOID}$$

$$\frac{H \vdash t_1 \leq t_2}{H \vdash_{rt} t_1 \leq t_2} \text{SUB_SUBRET_RTTYP}$$

3 Well-formed types

$H \vdash t$

$$\frac{}{H \vdash \mathbf{int}} \text{WF_TYPOKOKINT}$$

$$\frac{}{H \vdash \mathbf{bool}} \text{WF_TYPOKOKBOOL}$$

$$\frac{H \vdash_r \mathit{ref}}{H \vdash \mathit{ref}} \text{WF_TYPOKOKREFT}$$

$$\frac{H \vdash_r \mathit{ref}}{H \vdash \mathit{ref}^?} \text{WF_TYPOKOKREFTQ}$$

$H \vdash_r \mathit{ref}$

$$\frac{}{H \vdash_r \mathbf{string}} \text{WF_REFTOKOKSTRING}$$

$$\frac{H \vdash t}{H \vdash_r t[]} \text{WF_REFTOKOKARRAY}$$

$$\frac{\mathbf{struct} \ S\{\mathit{fields}\} \in H}{H \vdash_r S} \text{WF_REFTOKOKSTRUCT}$$

$$\frac{H \vdash t_1 \dots H \vdash t_n \quad H \vdash_{rt} \mathit{rt}}{H \vdash_r (t_1, \dots, t_n) \rightarrow \mathit{rt}} \text{WF_REFTOKOKFUNT}$$

$H \vdash_{rt} \mathit{rt}$

$$\frac{}{H \vdash_{rt} \mathbf{void}} \text{WF_RTYPOKVOIDOK}$$

$$\frac{H \vdash t}{H \vdash_{rt} t} \text{WF_RTYPOKRTYPOK}$$

4 Typing Rules

$\vdash bop_1, \dots, bop_i : t$

$\frac{}{\vdash +, *, -, <, >, >>, [\&], [!]: (\text{int}, \text{int}) \rightarrow \text{int}}$ TYP_INTOps

$\frac{}{\vdash <, <=, >, >=: (\text{int}, \text{int}) \rightarrow \text{bool}}$ TYP_CMPOps

$\frac{}{\vdash \&, |: (\text{bool}, \text{bool}) \rightarrow \text{bool}}$ TYP_BOOLOps

$\vdash uop : t$

$\frac{}{\vdash !: (\text{bool}) \rightarrow \text{bool}}$ TYP_LOGNOT

$\frac{}{\vdash \sim : (\text{int}) \rightarrow \text{int}}$ TYP_BITNEG

$\frac{}{\vdash -: (\text{int}) \rightarrow \text{int}}$ TYP_NEG

$H; G; L \vdash_{lhs} lhs : t; \text{assignable}$

$\frac{f \notin L \quad f: (t_1, \dots, t_n) \rightarrow rt \in G}{H; G; L \vdash_{lhs} f : (t_1, \dots, t_n) \rightarrow rt; \perp}$ TYP_FUNCTION_ID

$\frac{x \notin L \quad x: t \in G \quad x \text{ does not resolve to a global function}}{H; G; L \vdash_{lhs} x : t; \top}$ TYP_GLOBAL

$\frac{x: t \in L}{H; G; L \vdash_{lhs} x : t; \top}$ TYP_LOCAL

$\frac{H; G; L \vdash exp : S \quad \text{struct } S\{\text{fields}\} \in H \quad t x \in \text{fields}}{H; G; L \vdash_{lhs} exp.x : t; \top}$ TYP_FIELD

$\frac{H; G; L \vdash exp_1 : t[] \quad H; G; L \vdash exp_2 : \text{int}}{H; G; L \vdash_{lhs} exp_1[exp_2] : t; \top}$ TYP_INDEX

$H; G; L \vdash exp : t$

$\frac{H \vdash ref}{H; G; L \vdash ref \text{ null} : ref?}$ TYP_NULL

$\frac{}{H; G; L \vdash \text{true} : \text{bool}}$ TYP_BOOL_TRUE

$\frac{}{H; G; L \vdash \text{false} : \text{bool}}$ TYP_BOOL_FALSE

$\frac{}{H; G; L \vdash \text{integer} : \text{int}}$ TYP_INT

$\frac{}{H; G; L \vdash \text{string} : \text{string}}$ TYP_STRING

$\frac{H; G; L \vdash_{lhs} lhs : t; r}{H; G; L \vdash lhs : t}$ TYP_LHS

$$\begin{array}{c}
\frac{
\begin{array}{l}
H \vdash t \\
H;G;L \vdash \text{exp}_1 : t_1 \dots H;G;L \vdash \text{exp}_n : t_n \\
H \vdash t_1 \leq t \dots H \vdash t_n \leq t
\end{array}
}{
H;G;L \vdash \text{new } t[]\{\text{exp}_1, \dots, \text{exp}_n\} : t[]
} \text{TYP_CARR} \\
\\
\frac{
\begin{array}{l}
H \vdash t \\
H;G;L \vdash \text{exp}_1 : \text{int} \\
t \in \{\text{int}, \text{bool}, r?\}
\end{array}
}{
H;G;L \vdash \text{new } t[\text{exp}_1] : t[]
} \text{TYP_NEWARRAY} \\
\\
\frac{
\begin{array}{l}
H \vdash t \\
H;G;L \vdash \text{exp}_1 : \text{int} \\
x \notin L \quad H;G;L, x:\text{int} \vdash \text{exp}_2 : t' \quad H \vdash t' \leq t
\end{array}
}{
H;G;L \vdash \text{new } t[\text{exp}_1]\{x \rightarrow \text{exp}_2\} : t[]
} \text{TYP_NEWARRAYINIT} \\
\\
\frac{
H;G;L \vdash \text{exp} : t[]
}{
H;G;L \vdash \text{length}(\text{exp}) : \text{int}
} \text{TYP_LENGTH} \\
\\
\frac{
\begin{array}{l}
\text{struct } S\{t_1 x_1; \dots; t_n x_n\} \in H \\
H;G;L \vdash \text{exp}_1 : t'_1 \dots H;G;L \vdash \text{exp}_n : t'_n \\
H \vdash t'_1 \leq t_1 \dots H \vdash t'_n \leq t_n \\
\text{fields may be permuted under new}
\end{array}
}{
H;G;L \vdash \text{new } S\{x_1=\text{exp}_1; \dots; x_n=\text{exp}_n\} : S
} \text{TYP_STRUCTEX} \\
\\
\frac{
\begin{array}{l}
H;G;L \vdash \text{exp} : (t_1, \dots, t_n) \rightarrow t \\
H;G;L \vdash \text{exp}_1 : t'_1 \dots H;G;L \vdash \text{exp}_n : t'_n \\
H \vdash t'_1 \leq t_1 \dots H \vdash t'_n \leq t_n
\end{array}
}{
H;G;L \vdash \text{exp}(\text{exp}_1, \dots, \text{exp}_n) : t
} \text{TYP_CALL} \\
\\
\frac{
\begin{array}{l}
\vdash \text{bop} : (t_1, t_2) \rightarrow t \\
H;G;L \vdash \text{exp}_1 : t_1 \quad H;G;L \vdash \text{exp}_2 : t_2
\end{array}
}{
H;G;L \vdash \text{exp}_1 \text{ bop } \text{exp}_2 : t
} \text{TYP_BOP} \\
\\
\frac{
\begin{array}{l}
H;G;L \vdash \text{exp}_1 : t_1 \quad H;G;L \vdash \text{exp}_2 : t_2 \\
H \vdash t_1 \leq t_2 \quad H \vdash t_2 \leq t_1
\end{array}
}{
H;G;L \vdash \text{exp}_1 == \text{exp}_2 : \text{bool}
} \text{TYP_EQ} \\
\\
\frac{
\begin{array}{l}
H;G;L \vdash \text{exp}_1 : t_1 \quad H;G;L \vdash \text{exp}_2 : t_2 \\
H \vdash t_1 \leq t_2 \quad H \vdash t_2 \leq t_1
\end{array}
}{
H;G;L \vdash \text{exp}_1 != \text{exp}_2 : \text{bool}
} \text{TYP_NEQ} \\
\\
\frac{
\begin{array}{l}
\vdash \text{uop} : (t) \rightarrow t \quad H;G;L \vdash \text{exp} : t
\end{array}
}{
H;G;L \vdash \text{uop } \text{exp} : t
} \text{TYP_UOP}
\end{array}$$

$$\boxed{H;G;L_1 \vdash vdecl \Rightarrow L_2}$$

$$\frac{H;G;L \vdash exp : t \quad x \notin L}{H;G;L \vdash \text{var } x = exp \Rightarrow L, x:t} \text{ TYP_DECL}$$

$$\boxed{H;G;L_0 \vdash vdecls \Rightarrow L_i}$$

$$\frac{H;G;L_0 \vdash vdecl_1 \Rightarrow L_1 \quad \dots \quad H;G;L_{n-1} \vdash vdecl_i \Rightarrow L_n}{H;G;L_0 \vdash vdecl_1, \dots, vdecl_n \Rightarrow L_n} \text{ TYP_VDECLS}$$

$$\boxed{H;G;L_1;rt \vdash stmt \Rightarrow L_2;returns}$$

$$\frac{H;G;L \vdash_{lhs} lhs : t; \top \quad H;G;L \vdash exp : t' \quad H \vdash t' \leq t}{H;G;L;rt \vdash lhs = exp; \Rightarrow L; \perp} \text{ TYP_ASSN}$$

$$\frac{H;G;L_1 \vdash vdecl \Rightarrow L_2}{H;G;L_1;rt \vdash vdecl; \Rightarrow L_2; \perp} \text{ TYP_STMTDECL}$$

$$\frac{H;G;L \vdash exp : (t_1, \dots, t_n) \rightarrow \text{void} \quad H;G;L \vdash exp_1 : t'_1 \dots H;G;L \vdash exp_n : t'_n \quad H \vdash t'_1 \leq t_1 \dots H \vdash t'_n \leq t_n}{H;G;L;rt \vdash exp(exp_1, \dots, exp_n); \Rightarrow L; \perp} \text{ TYP_SCALL}$$

$$\frac{H;G;L \vdash exp : \text{bool} \quad H;G;L;rt \vdash block_1; r_1 \quad H;G;L;rt \vdash block_2; r_2}{H;G;L;rt \vdash \text{if}(exp) block_1 \text{ else } block_2 \Rightarrow L; r_1 \wedge r_2} \text{ TYP_IF}$$

$$\frac{x \notin L \quad H;G;L \vdash exp : ref' ? \quad H \vdash ref' \leq ref \quad H;G;L, x:ref; rt \vdash block_1; r_1 \quad H;G;L;rt \vdash block_2; r_2}{H;G;L;rt \vdash \text{if?}(ref \ x = exp) block_1 \text{ else } block_2 \Rightarrow L; r_1 \wedge r_2} \text{ TYP_IFQ}$$

$$\frac{H;G;L \vdash exp : \text{bool} \quad H;G;L;rt \vdash block; r}{H;G;L;rt \vdash \text{while}(exp) block \Rightarrow L; \perp} \text{ TYP_WHILE}$$

$$\frac{H;G;L_1 \vdash vdecls \Rightarrow L_2 \quad H;G;L_2 \vdash exp : \text{bool} \quad H;G;L_2;rt \vdash stmt \Rightarrow L_3; \perp \quad H;G;L_2;rt \vdash block; r}{H;G;L_1;rt \vdash \text{for}(vdecls; exp_{opt}; stmt_{opt}) block \Rightarrow L_1; \perp} \text{ TYP_FOR}$$

$$\frac{H;G;L \vdash exp : t' \quad H \vdash t' \leq t}{H;G;L;rt \vdash \text{return } exp; \Rightarrow L; \top} \text{ TYP_RETT}$$

$$\frac{}{H;G;L; \text{void} \vdash \text{return}; \Rightarrow L; \top} \text{ TYP_RETVOID}$$

$H;G;L;rt \vdash \text{block};\text{returns}$

$$\frac{H;G;L_0;rt \vdash_{ss} \text{stmt}_1 .. \text{stmt}_n \Rightarrow L_n;r}{H;G;L_0;rt \vdash \{\text{stmt}_1 .. \text{stmt}_n\};r} \text{ TYP_BLOCK}$$

$H;G;L_0;rt \vdash_{ss} \text{stmt}_1 .. \text{stmt}_n \Rightarrow L_n;\text{returns}$

$$\frac{\begin{array}{l} H;G;L_0;rt \vdash \text{stmt}_1 \Rightarrow L_1;\perp \\ \dots \\ H;G;L_{n-2};rt \vdash \text{stmt}_{n-1} \Rightarrow L_{n-1};\perp \\ H;G;L_{n-1};rt \vdash \text{stmt}_n \Rightarrow L_n;r \end{array}}{H;G;L_0;rt \vdash_{ss} \text{stmt}_1 .. \text{stmt}_{n-1} \text{stmt}_n \Rightarrow L_n;r} \text{ TYP_STMTS}$$

$H;G \vdash_s \text{tdecl}$

$$\frac{H \vdash t_1 .. H \vdash t_i \quad x_1 .. x_i \text{ distinct}}{H;G \vdash_s \text{struct } S\{t_1 x_1; .. ; t_i x_i\}} \text{ TYP_TDECLOK}$$

$H;G \vdash_f \text{fdecl}$

$$\frac{H;G;x_1:t_1, .., x_i:t_i;rt \vdash \text{block};\top \quad x_1 .. x_i \text{ distinct}}{H;G \vdash_f rt f(t_1 x_1, .., t_i x_i) \text{ block}} \text{ TYP_FDECLOK}$$

$H;G \vdash \text{prog}$

$$\frac{}{H;G \vdash \epsilon} \text{ TYP_EMPTY}$$

$$\frac{H;G \vdash \text{prog}}{H;G \vdash \text{gdecl prog}} \text{ TYP_DGDECL}$$

$$\frac{H;G \vdash_f \text{fdecl} \quad H;G \vdash \text{prog}}{H;G \vdash \text{fdecl prog}} \text{ TYP_DFDECL}$$

$$\frac{H;G \vdash_s \text{tdecl} \quad H;G \vdash \text{prog}}{H;G \vdash \text{tdecl prog}} \text{ TYP_DTDECL}$$

$H;G_1 \vdash_g \text{prog} \Rightarrow G_2$

$$\frac{}{H;G \vdash_g \epsilon \Rightarrow G} \text{ TYP_GEMPTY}$$

$$\frac{H;G_1 \vdash_g \text{prog} \Rightarrow G_2}{H;G_1 \vdash_g \text{tdecl prog} \Rightarrow G_2} \text{ TYP_GTDECL}$$

$$\frac{\begin{array}{l} H;G_1;\cdot \vdash \text{gexp} : t \\ x \notin G_1 \quad H;G_1,x:t \vdash_g \text{prog} \Rightarrow G_2 \end{array}}{H;G_1 \vdash_g \text{global } x = \text{gexp}; \text{prog} \Rightarrow G_2} \text{ TYP_GGDECL}$$

$$\frac{H;G_1 \vdash_g \text{prog} \Rightarrow G_2}{H;G_1 \vdash_g \text{fdecl prog} \Rightarrow G_2} \text{ TYP_GFDECL}$$

$H \vdash \text{fdecl} \Rightarrow \text{id}:t$

$$\frac{H \vdash_{rt} rt \quad H \vdash t_1 .. H \vdash t_n}{H \vdash_{rt} f(t_1 x_1, .., t_n x_n) \text{ block} \Rightarrow f:(t_1, .., t_n) \rightarrow rt} \text{ TYP_FTYP}$$

$$\boxed{H; G_1 \vdash_f \text{prog} \Rightarrow G_2}$$

$$\frac{}{H; G \vdash_f \epsilon \Rightarrow G} \text{ TYP_FEMPTY}$$

$$\frac{H; G_1 \vdash_f \text{prog} \Rightarrow G_2}{H; G_1 \vdash_f \text{tdecl prog} \Rightarrow G_2} \text{ TYP_FTDECL}$$

$$\frac{H; G_1 \vdash_f \text{prog} \Rightarrow G_2}{H; G_1 \vdash_f \text{gdecl prog} \Rightarrow G_2} \text{ TYP_FGDECL}$$

$$\frac{H \vdash \text{fdecl} \Rightarrow f:t \quad f \notin G_1 \quad H; G_1, f:t \vdash_f \text{prog} \Rightarrow G_2}{H; G_1 \vdash_f \text{fdecl prog} \Rightarrow G_2} \text{ TYP_FFDECL}$$

$$\boxed{H_1 \vdash_s \text{prog} \Rightarrow H_2}$$

$$\frac{}{H \vdash_s \epsilon \Rightarrow H} \text{ TYP_SEEMPTY}$$

$$\frac{S \notin H_1 \quad H_1, \text{struct } S\{ \text{fields} \} \vdash_s \text{prog} \Rightarrow H_2}{H_1 \vdash_s \text{struct } S\{ \text{fields} \} \text{ prog} \Rightarrow H_2} \text{ TYP_STDECL}$$

$$\frac{H_1 \vdash_s \text{prog} \Rightarrow H_2}{H_1 \vdash_s \text{gdecl prog} \Rightarrow H_2} \text{ TYP_SGDECL}$$

$$\frac{H_1 \vdash_s \text{prog} \Rightarrow H_2}{H_1 \vdash_s \text{fdecl prog} \Rightarrow H_2} \text{ TYP_SFDECL}$$

$$\boxed{\vdash \text{prog}}$$

$$\frac{\cdot \vdash_s \text{prog} \Rightarrow H \quad H; G_0 \vdash_f \text{prog} \Rightarrow G_1 \quad H; G_1 \vdash_g \text{prog} \Rightarrow G_2 \quad H; G_2 \vdash \text{prog}}{\vdash \text{prog}} \text{ TYP_PROG}$$

Notes:

- The context G_0 mentioned in the rule for typechecking a complete, top-level program is the “initial context” which should contain bindings for all of the OAT built-in functions.
- The type system processes the program in several passes: (1) collect up all the structure type definitions and make sure their names don’t clash using the \vdash_s rules, (2) add all the function identifiers and their types to the global context using the \vdash_f rules, again ensuring no name clashes, (3) typecheck the global value declarations and add them to the context using the \vdash_g rules, and (4) process all the declarations one more time to examine all the struct fields to make sure their types are well formed and to typecheck the function bodies.

The rules therefore allow the types of structs to be mutually recursive, and for global values to mention function pointers as constants.

- We use \perp to indicate that a statement might not return, and \top to indicate that a statement definitely returns. When typechecking the list of statements that make up a block, only the last statement is allowed to definitely return (all the others must possibly not return). Note that `TYP_FDECLOK` requires that the block making up a function body definitely return.