

Oat v. 2 Language Specification

CS516

April 4, 2023

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 are not part of the abstract syntax, but are included for parsing purposes (e.g. parentheses) or for use in explaining the typing judgments. Note that left-hand-sides (*lhs*) and global expressions (*gexp*) are just a subsets of expressions.

<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>	
	<code>S</code>	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>	
	<code><i>t</i></code>	
<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</code> <i>id</i> = <i>gexp</i> ;	global value declarations
<i>arg</i>	$::=$ <i>t id</i>	arg
<i>args</i>	$::=$ <i>arg</i> ₁ , .., <i>arg</i> _{<i>n</i>}	args
<i>fdecl</i>	$::=$ <i>rt id</i> (<i>args</i>) <i>block</i>	function declaration
<i>field</i>	$::=$ <i>t x</i>	field declarations
<i>fields</i>	$::=$ <i>field</i> ₁ ; ..; <i>field</i> _{<i>n</i>} <i>fields</i> ; <i>t</i> _{<i>n</i>+1} <i>x</i> _{<i>n</i>+1} ; ..; <i>t</i> _{<i>m</i>} <i>x</i> _{<i>m</i>}	fields
<i>tdecl</i>	$::=$ <code>struct</code> <i>S</i> { <i>fields</i> }	struct declaration
<i>exp</i>	$::=$ <code>ref</code> <code>null</code> <code>true</code> <code>false</code> <i>integer</i> <i>string</i> <i>id</i> <code>new</code> <i>t</i> [] { <i>exp</i> ₁ , .., <i>exp</i> _{<i>n</i>} } <code>new</code> <i>t</i> [<i>exp</i>] <code>new</code> <i>t</i> [<i>exp</i> ₁] { <i>id</i> -> <i>exp</i> ₂ } <i>exp</i> ₁ [<i>exp</i> ₂] <code>length</code> (<i>exp</i>) <code>new</code> <i>S</i> { <i>x</i> ₁ = <i>exp</i> ₁ ; ..; <i>x</i> _{<i>n</i>} = <i>exp</i> _{<i>n</i>} } <i>exp</i> . <i>id</i> <i>exp</i> (<i>exp</i> ₁ , .., <i>exp</i> _{<i>n</i>}) <i>uop</i> <i>exp</i> <i>exp</i> ₁ <i>bop</i> <i>exp</i> ₂ (<i>exp</i>)	expressions type-annotated null value 64-bit integer literals C-style strings global or local variable array literal value array with default initializer array with explicit initializer array index built-in polymorphic length operation struct initialization field projection function call unary operation binary operation
<i>lhs</i>	$::=$ <i>id</i> <i>exp</i> ₁ [<i>exp</i> ₂] <i>exp</i> . <i>id</i>	left-hand-sides for assignment
<i>gexp</i>	$::=$ <i>integer</i>	global initializers 64-bit integer literals

	$\mid$ string $\mid$ $\text{ref } \text{null}$ $\mid$ true $\mid$ false $\mid$ $\text{new } t[]\{\text{gexp}_1, \dots, \text{gexp}_n\}$ $\mid$ $\text{new } S\{x_1=\text{gexp}_1; \dots; x_n=\text{gexp}_n\}$ $\mid$ id	C-style strings
block	$::=$ $\mid$ $\{\text{stmt}_1 \dots \text{stmt}_n\}$	blocks
vdecl	$::=$ $\mid$ $\text{var } \text{id} = \text{exp}$	local variable declarations
stmt	$::=$ $\mid$ $\text{lhs} = \text{exp};$ $\mid$ $\text{vdecl};$ $\mid$ $\text{return } \text{exp};$ $\mid$ $\text{return};$ $\mid$ $\text{exp}(\text{exp}_1, \dots, \text{exp}_n);$ $\mid$ if_stmt $\mid$ $\text{for}(\text{vdecls}; \text{exp}_{\text{opt}}; \text{stmt}_{\text{opt}}) \text{ block}$ $\mid$ $\text{while}(\text{exp}) \text{ block}$	statements assignment statement variable declaration return with value return (void) call a void-returning function conditionals
if_stmt	$::=$ $\mid$ $\text{if}(\text{exp}) \text{ block } \text{else_stmt}$ $\mid$ $\text{if?}(\text{ref } \text{id} = \text{exp}) \text{ block } \text{else_stmt}$	if statements standard boolean if possibly-null checked downcast
else_stmt	$::=$ $\mid$ ϵ $\mid$ $\text{else } \text{block}$ $\mid$ $\text{else } \text{if_stmt}$	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

$H \vdash t$

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

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

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

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

$H \vdash_r \text{ref}$

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

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

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

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

$H \vdash_{rt} rt$

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

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

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

$H; G; L \vdash \text{exp} : t$

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

$\frac{id : t \in L}{H; G; L \vdash id : t}$ TYP_LOCAL

$\frac{id \notin L \quad id : t \in G}{H; G; L \vdash id : t}$ TYP_GLOBAL

$\frac{\begin{array}{l} H \vdash t \\ H; G; L \vdash \text{exp}_1 : t_1 \dots H; G; L \vdash \text{exp}_n : t_n \\ H \vdash t_1 \leq t \dots H \vdash t_n \leq t \end{array}}{H; G; L \vdash \text{new } t[]\{\text{exp}_1, \dots, \text{exp}_n\} : t[]}$ TYP_CARR

$\frac{\begin{array}{l} H \vdash t \\ H; G; L \vdash \text{exp}_1 : \text{int} \\ t \in \{\text{int}, \text{bool}, r?\} \end{array}}{H; G; L \vdash \text{new } t[\text{exp}_1] : t[]}$ TYP_NEWARRAY

$\frac{\begin{array}{l} H \vdash t \\ H; G; L \vdash \text{exp}_1 : \text{int} \\ x \notin L \quad H; G; L, x : \text{int} \vdash \text{exp}_2 : t' \quad H \vdash t' \leq t \end{array}}{H; G; L \vdash \text{new } t[\text{exp}_1]\{x \rightarrow \text{exp}_2\} : t[]}$ TYP_NEWARRAYINT

$$\begin{array}{c}
\frac{H;G;L \vdash \text{exp}_1 : t[] \quad H;G;L \vdash \text{exp}_2 : \text{int}}{H;G;L \vdash \text{exp}_1[\text{exp}_2] : t} \quad \text{TYP_INDEX} \\
\\
\frac{H;G;L \vdash \text{exp} : t[]}{H;G;L \vdash \text{length}(\text{exp}) : \text{int}} \quad \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} \quad \text{TYP_STRUCTEX} \\
\\
\frac{\begin{array}{l} H;G;L \vdash \text{exp} : S \\ \text{struct } S\{\text{fields}\} \in H \quad t x \in \text{fields} \end{array}}{H;G;L \vdash \text{exp}.x : t} \quad \text{TYP_FIELD} \\
\\
\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} \quad \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} \quad \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}} \quad \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}} \quad \text{TYP_NEQ} \\
\\
\frac{\vdash \text{uop} : (t) \rightarrow t \quad H;G;L \vdash \text{exp} : t}{H;G;L \vdash \text{uop exp} : t} \quad \text{TYP_UOP}
\end{array}$$

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

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

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

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

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

$$\frac{\begin{array}{l} \text{lhs not a global function id} \\ H;G;L \vdash \text{lhs} : t \\ H;G;L \vdash \text{exp} : t' \\ H \vdash t' \leq t \end{array}}{H;G;L;rt \vdash \text{lhs} = \text{exp}; \Rightarrow L; \perp} \quad \text{TYP_ASSN}$$

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

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

$$\frac{\begin{array}{l} H;G;L \vdash \text{exp} : \text{bool} \\ H;G;L;rt \vdash \text{block}_1; r_1 \\ H;G;L;rt \vdash \text{block}_2; r_2 \end{array}}{H;G;L;rt \vdash \text{if}(\text{exp}) \text{block}_1 \text{ else } \text{block}_2 \Rightarrow L; r_1 \wedge r_2} \quad \text{TYP_IF}$$

$$\frac{\begin{array}{l} H;G;L \vdash \text{exp} : \text{ref}? \\ H \vdash \text{ref}' \leq \text{ref} \\ H;G;L, x:\text{ref};rt \vdash \text{block}_1; r_1 \quad H;G;L;rt \vdash \text{block}_2; r_2 \end{array}}{H;G;L;rt \vdash \text{if?}(\text{ref } x = \text{exp}) \text{block}_1 \text{ else } \text{block}_2 \Rightarrow L; r_1 \wedge r_2} \quad \text{TYP_IFQ}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\overline{H;G \vdash \epsilon} \quad \text{TYP_EMPTY}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\boxed{\vdash prog}$$

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

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) type-check 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.