Oat Language Specification

CIS341 – Steve Zdancewic

March 30, 2020

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.

t	::=			types
	|	`int`		
	|	`bool`		
	|	*ref*`?`		nullable references
	|	*ref*		non-null references
ref	::=			reference types
	|	`string`		
	|	*S*		named (mutable) record type
	|	*t*`[]`		arrays
	|	`(`t_0`,` .. `,` t_n`)` `->` *rt*		function pointer
	|	`(`*ref*`)`	M	
rt	::=			return types
	|	`void`		
	|	*t*		

prog	::=		prog
	|	ϵ	
	|	*decl prog*	
decl	::=		global declarations
	|	*gdecl*	
	|	*fdecl*	
	|	*tdecl*	

gdecl	::=			global value declarations
	|	`global` *id* = *gexp* ;		
arg	::=			arg
	|	*t id*		
args	::=			args
	|	arg_1 , .. , arg_n		
fdecl	::=			function declaration
	|	*rt id* (*args*) *block*		
field	::=			field declarations
	|	*t x*		
fields	::=			fields
	|	$field_1$; .. ; $field_n$		
	|	*fields* ; t_{n+1} x_{n+1} ; .. ; t_m x_m	M	
tdecl	::=			struct declaration
	|	`struct` *S*{ *fields* }		

exp	::=			expressions
	|	*ref* `null`		type-annotated null value
	|	`true`		
	|	`false`		
	|	*integer*		64-bit integer literals
	|	*string*		C-style strings
	|	*id*		global or local variable
	|	`new` *t*[]{ exp_1 , .. , exp_n }		array literal value
	|	`new` *t* [exp_1] { *id* -> exp_2 }		array with "initializer"
	|	exp_1 [exp_2]		array index
	|	`length` (*exp*)		built-in polymorphic length operation
	|	`new` *S*{ x_1 = exp_1 ; .. ; x_n = exp_n }		struct initialization
	|	*exp* . *id*		field projection
	|	*exp* (exp_1 , .. , exp_n)		function call
	|	*uop exp*		unary operation
	|	exp_1 *bop* exp_2		binary operation
	|	(*exp*)	M	
lhs	::=			left-hand-sides for assignment
	|	*id*		
	|	exp_1 [exp_2]		
	|	*exp* . *id*		
gexp	::=			global initializers

	|	*integer*	64-bit integer literals
	|	*string*	C-style strings
	|	*ref* `null`	
	|	`true`	
	|	`false`	
	|	`new` *t*[]{$gexp_1$, .. ,$gexp_n$}	
	|	`new` *S*{x_1=$gexp_1$; .. ; x_n=$gexp_n$}	
	|	*id*	
block	::=		blocks
	|	{$stmt_1$ .. $stmt_n$}	
vdecl	::=		local variable declarations
	|	`var` *id* = *exp*	
stmt	::=		statements
	|	*lhs* = *exp* ;	assignment statement
	|	*vdecl* ;	variable declaration
	|	`return` *exp* ;	return with value
	|	`return`;	return (void)
	|	*exp* (exp_1 , .. , exp_n) ;	call a void-returning function
	|	*if_stmt*	conditionals
	|	`for`(*vdecls* ; exp_{opt} ; $stmt_{opt}$) *block*	
	|	`while`(*exp*) *block*	
if_stmt	::=		if statements
	|	`if`(*exp*) *block else_stmt*	standard boolean if
	|	`if`?(*ref id* = *exp*) *block else_stmt*	possibly-null checked downcast
else_stmt	::=		else
	|	ϵ	
	|	`else` *block*	
	|	`else` *if_stmt*	

```
bop    ::=              (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)

uop    ::=              unary operations
         |   -
         |   !
         |   ~
```

2 Subtyping Rules

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

$$\frac{}{H \vdash \texttt{int} \ \leq \ \texttt{int}} \quad \textsc{sub\_sub\_int}$$

$$\frac{}{H \vdash \texttt{bool} \ \leq \ \texttt{bool}} \quad \textsc{sub\_sub\_bool}$$

$$\frac{H \vdash_r \mathit{ref}_1 \ \leq \ \mathit{ref}_2}{H \vdash \mathit{ref}_1? \ \leq \ \mathit{ref}_2?} \quad \textsc{sub\_sub\_nref}$$

$$\frac{H \vdash_r \mathit{ref}_1 \ \leq \ \mathit{ref}_2}{H \vdash \mathit{ref}_1 \ \leq \ \mathit{ref}_2} \quad \textsc{sub\_sub\_ref}$$

$$\frac{H \vdash_r \mathit{ref}_1 \ \leq \ \mathit{ref}_2}{H \vdash \mathit{ref}_1 \ \leq \ \mathit{ref}_2?} \quad \textsc{sub\_sub\_nrref}$$

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

$$\frac{}{H \vdash_r \texttt{string} \ \leq \ \texttt{string}} \quad \textsc{sub\_subrstring}$$

$$\frac{}{H \vdash_r t\texttt{[]} \ \leq \ t\texttt{[]}} \quad \textsc{sub\_subrarray}$$

$$\frac{\texttt{struct}\ S_1\texttt{\{}\ t_1\ x_1\texttt{;}\ ..\ \texttt{;}t_n\ x_n\texttt{;}t_{n+1}\ x_{n+1}\texttt{;}..\texttt{;}t_m\ x_m\ \texttt{\}} \ \in H \quad \texttt{struct}\ S_2\texttt{\{}\ t_1\ x_1\texttt{;}\ ..\ \texttt{;}t_n\ x_n\ \texttt{\}} \ \in H}{H \vdash_r S_1 \ \leq \ S_2} \quad \textsc{sub\_subrstruct}$$

$$\frac{H \vdash t'_1 \ \leq \ t_1 \quad .. \quad H \vdash t'_n \ \leq \ t_n \quad H \vdash_{rt} rt_1 \leq rt_2}{H \vdash_r \texttt{(}t_1, .., t_n\texttt{)} \texttt{ -> } rt_1 \ \leq \ \texttt{(}t'_1, .., t'_n\texttt{)} \texttt{ -> } rt_2} \quad \textsc{sub\_subrfunt}$$

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

$$\frac{}{H \vdash_{rt} \texttt{void} \leq \texttt{void}} \quad \textsc{sub\_subretsvoid}$$

$$\frac{H \vdash t_1 \ \leq \ t_2}{H \vdash_{rt} t_1 \leq t_2} \quad \textsc{sub\_subretrttyp}$$

3 Well-formed types

$\boxed{H \vdash t}$

$$\frac{}{H \vdash \texttt{int}} \quad \text{WF\_TYPOKOKINT}$$

$$\frac{}{H \vdash \texttt{bool}} \quad \text{WF\_TYPOKOKBOOL}$$

$$\frac{H \vdash_r \mathit{ref}}{H \vdash \mathit{ref}} \quad \text{WF\_TYPOKOKREFT}$$

$$\frac{H \vdash_r \mathit{ref}}{H \vdash \mathit{ref}\,\texttt{?}} \quad \text{WF\_TYPOKOKREFTQ}$$

$\boxed{H \vdash_r \mathit{ref}}$

$$\frac{}{H \vdash_r \texttt{string}} \quad \text{WF\_REFTOKOKSTRING}$$

$$\frac{H \vdash t}{H \vdash_r t\,\texttt{[]}} \quad \text{WF\_REFTOKOKARRAY}$$

$$\frac{\texttt{struct}\ S\texttt{\{}\ \mathit{fields}\ \texttt{\}} \quad \in H}{H \vdash_r S} \quad \text{WF\_REFTOKOKSTRUCT}$$

$$\frac{H \vdash t_1 \quad .. \quad H \vdash t_n \quad H \vdash_{rt} rt}{H \vdash_r \texttt{(}t_1\texttt{,}\ ..\ \texttt{,}t_n\texttt{)}\ \texttt{->}\ rt} \quad \text{WF\_REFTOKOKFUNT}$$

$\boxed{H \vdash_{rt} rt}$

$$\frac{}{H \vdash_{rt} \texttt{void}} \quad \text{WF\_RTYPOKVOIDOK}$$

$$\frac{H \vdash t}{H \vdash_{rt} t} \quad \text{WF\_RTYPOKRTYPOK}$$

4 Typing Rules

$\boxed{\vdash bop_1, .., bop_i : t}$

$$\frac{}{\vdash \texttt{+},\texttt{*},\texttt{-},\texttt{<<},\texttt{>>},\texttt{>>>},\texttt{[\&]},\texttt{[|]} : (\texttt{int},\texttt{int})\ \texttt{->}\ \texttt{int}}\ \text{TYP\_INTOPS}$$

$$\frac{}{\vdash \texttt{<},\texttt{<=},\texttt{>},\texttt{>=} : (\texttt{int},\texttt{int})\ \texttt{->}\ \texttt{bool}}\ \text{TYP\_CMPOPS}$$

$$\frac{}{\vdash \texttt{\&},\texttt{|} : (\texttt{bool},\texttt{bool})\ \texttt{->}\ \texttt{bool}}\ \text{TYP\_BOOLOPS}$$

$\boxed{\vdash uop : t}$

$$\frac{}{\vdash \texttt{!} : (\texttt{bool})\ \texttt{->}\ \texttt{bool}}\ \text{TYP\_LOGNOT}$$

$$\frac{}{\vdash \texttt{\~{}} : (\texttt{int})\ \texttt{->}\ \texttt{int}}\ \text{TYP\_BITNEG}$$

$$\frac{}{\vdash \texttt{-} : (\texttt{int})\ \texttt{->}\ \texttt{int}}\ \text{TYP\_NEG}$$

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

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

$$\frac{}{H;G;L \vdash\ \texttt{true}\ :\ \texttt{bool}}\ \text{TYP\_BOOL\_TRUE}$$

$$\frac{}{H;G;L \vdash\ \texttt{false}\ :\ \texttt{bool}}\ \text{TYP\_BOOL\_FALSE}$$

$$\frac{}{H;G;L \vdash\ integer\ :\ \texttt{int}}\ \text{TYP\_INT}$$

$$\frac{}{H;G;L \vdash\ string\ :\ \texttt{string}}\ \text{TYP\_STRING}$$

$$\frac{id : t \in L}{H;G;L \vdash\ id\ :\ t}\ \text{TYP\_LOCAL}$$

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

$$\frac{\begin{array}{l} H \vdash t \\ H;G;L \vdash\ exp_1\ :\ t_1 \quad .. \quad H;G;L \vdash\ exp_n\ :\ t_n \\ H \vdash t_1\ \leq\ t \quad .. \quad H \vdash t_n\ \leq\ t \end{array}}{H;G;L \vdash\ \texttt{new}\ t[]\{exp_1, .., exp_n\}\ \ :\ t\texttt{[]}}\ \text{TYP\_CARR}$$

$$\frac{\begin{array}{l} H \vdash t \\ H;G;L \vdash\ exp_1\ :\ \texttt{int} \\ x \notin L \quad H;G;L,x:\texttt{int} \vdash\ exp_2\ :\ t' \quad H \vdash t'\ \leq\ t \end{array}}{H;G;L \vdash\ \texttt{new}\ t\texttt{[}exp_1\texttt{]\{}x\ \texttt{->}\ exp_2\texttt{\}}\ \ :\ t\texttt{[]}}\ \text{TYP\_NEWARRAY}$$

$$\frac{H;G;L \vdash\ exp_1\ :\ t\texttt{[]} \quad H;G;L \vdash\ exp_2\ :\ \texttt{int}}{H;G;L \vdash\ exp_1\texttt{[}exp_2\texttt{]}\ :\ t}\ \text{TYP\_INDEX}$$

$$\frac{H;G;L \vdash\ exp\ :\ t\texttt{[]}}{H;G;L \vdash\ \texttt{length}\,(exp)\ :\ \texttt{int}}\ \text{TYP\_LENGTH}$$

$$\frac{\begin{array}{l}\texttt{struct}\ S\{\ t_1\ x_1\,;\,..\,;t_n\ x_n\ \}\ \ \in H\\ H;G;L \vdash\ exp_1\ :\ t'_1\quad ..\quad H;G;L\vdash\ exp_n\ :\ t'_n\\ H\vdash t'_1\ \le\ t_1\quad ..\quad H\vdash t'_n\ \le\ t_n\\ \textit{fields may be permuted under } \texttt{new}\end{array}}{H;G;L\vdash\ \texttt{new}\,S\{x_1\texttt{=}exp_1\,;\,..\,;x_n\texttt{=}exp_n\}\quad :\ S}\quad \text{TYP\_STRUCTEX}$$

$$\frac{\begin{array}{l}H;G;L\vdash\ exp\ :\ S\\ \texttt{struct}\ S\{\,fields\ \}\ \ \in H\quad t\ x\in fields\end{array}}{H;G;L\vdash\ exp\,.\,x\ :\ t}\quad \text{TYP\_FIELD}$$

$$\frac{\begin{array}{l}H;G;L\vdash\ exp\ :\ (t_1,\,..\,,t_n)\ \texttt{->}\ t\\ H;G;L\vdash\ exp_1\ :\ t'_1\quad ..\quad H;G;L\vdash\ exp_n\ :\ t'_n\\ H\vdash t'_1\ \le\ t_1\quad ..\quad H\vdash t'_n\ \le\ t_n\end{array}}{H;G;L\vdash\ exp\,(exp_1\,,\,..\,,exp_n)\ :\ t}\quad \text{TYP\_CALL}$$

$$\frac{\begin{array}{l}\vdash bop\,:(t_1,t_2)\ \texttt{->}\ t\\ H;G;L\vdash\ exp_1\ :\ t_1\quad H;G;L\vdash\ exp_2\ :\ t_2\end{array}}{H;G;L\vdash\ exp_1\,bop\,exp_2\ :\ t}\quad \text{TYP\_BOP}$$

$$\frac{\begin{array}{l}H;G;L\vdash\ exp_1\ :\ t_1\quad H;G;L\vdash\ exp_2\ :\ t_2\\ H\vdash t_1\ \le\ t_2\quad H\vdash t_2\ \le\ t_1\end{array}}{H;G;L\vdash\ exp_1\,\texttt{==}\,exp_2\ :\ \texttt{bool}}\quad \text{TYP\_EQ}$$

$$\frac{\begin{array}{l}H;G;L\vdash\ exp_1\ :\ t_1\quad H;G;L\vdash\ exp_2\ :\ t_2\\ H\vdash t_1\ \le\ t_2\quad H\vdash t_2\ \le\ t_1\end{array}}{H;G;L\vdash\ exp_1\,\texttt{!=}\,exp_2\ :\ \texttt{bool}}\quad \text{TYP\_NEQ}$$

$$\frac{\vdash uop\,:(t)\ \texttt{->}\ t\quad H;G;L\vdash\ exp\ :\ t}{H;G;L\vdash\ uop\,exp\ :\ t}\quad \text{TYP\_UOP}$$

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

$$\frac{H;G;L \vdash \ \mathit{exp} \ : \ t \quad x \notin L}{H;G;L \vdash \texttt{var}\ x = \mathit{exp} \Rightarrow L, x\!:\!t} \ \textsc{typ\_decl}$$

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

$$\frac{H;G;L_0 \vdash \mathit{vdecl}_1 \Rightarrow L_1 \quad \ldots \quad H;G;L_{n-1} \vdash \mathit{vdecl}_i \Rightarrow L_n}{H;G;L_0 \vdash \mathit{vdecl}_1 , .. , \mathit{vdecl}_n \Rightarrow L_n} \ \textsc{typ\_vdecls}$$

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

$$\frac{\begin{array}{l} G;L \vdash \mathit{lhs\ in\ L\ or\ lhs\ not\ a\ global\ function\ id} \\ H;G;L \vdash \ \mathit{lhs} \ : \ t \\ H;G;L \vdash \ \mathit{exp} \ : \ t' \\ H \vdash t' \ \leq \ t \end{array}}{H;G;L;rt \vdash \ \mathit{lhs} = \mathit{exp}; \ \Rightarrow \ L;\bot} \ \textsc{typ\_assn}$$

$$\frac{H;G;L_1 \vdash \mathit{vdecl} \Rightarrow L_2}{H;G;L_1;rt \vdash \ \mathit{vdecl}; \ \Rightarrow \ L_2;\bot} \ \textsc{typ\_stmtdecl}$$

$$\frac{\begin{array}{l} H;G;L \vdash \ \mathit{exp} \ : \ (t_1, .., t_n) \ \texttt{->}\ \texttt{void} \\ H;G;L \vdash \ \mathit{exp}_1 \ : \ t'_1 \quad .. \quad H;G;L \vdash \ \mathit{exp}_n \ : \ t'_n \\ H \vdash t'_1 \ \leq \ t_1 \quad .. \quad H \vdash t'_n \ \leq \ t_n \end{array}}{H;G;L;rt \vdash \ \mathit{exp}(\mathit{exp}_1 , .. , \mathit{exp}_n) ; \ \Rightarrow \ L;\bot} \ \textsc{typ\_scall}$$

$$\frac{\begin{array}{l} H;G;L \vdash \ \mathit{exp} \ : \ \texttt{bool} \\ H;G;L;rt \vdash \mathit{block}_1 ; r_1 \\ H;G;L;rt \vdash \mathit{block}_2 ; r_2 \end{array}}{H;G;L;rt \vdash \ \texttt{if}(\mathit{exp}) \ \mathit{block}_1 \ \texttt{else} \ \mathit{block}_2 \ \Rightarrow \ L; r_1 \wedge r_2} \ \textsc{typ\_if}$$

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

$$\frac{\begin{array}{l} H;G;L \vdash \ \mathit{exp} \ : \ \texttt{bool} \\ H;G;L;rt \vdash \mathit{block} ; r \end{array}}{H;G;L;rt \vdash \ \texttt{while}(\mathit{exp}) \ \mathit{block} \ \Rightarrow \ L;\bot} \ \textsc{typ\_while}$$

$$\frac{\begin{array}{l} H;G;L_1 \vdash \mathit{vdecls} \Rightarrow L_2 \\ H;G;L_2 \vdash \ \mathit{exp} \ : \ \texttt{bool} \\ H;G;L_2;rt \vdash \ \mathit{stmt} \ \Rightarrow \ L_3;\bot \\ H;G;L_2;rt \vdash \mathit{block} ; r \end{array}}{H;G;L_1;rt \vdash \ \texttt{for}(\mathit{vdecls}; \ \mathit{exp}_{opt} ; \ \mathit{stmt}_{opt}) \ \mathit{block} \ \Rightarrow \ L_1;\bot} \ \textsc{typ\_for}$$

$$\frac{H;G;L \vdash \ \mathit{exp} \ : \ t' \quad H \vdash t' \ \leq \ t}{H;G;L;t \vdash \ \texttt{return}\ \mathit{exp}; \ \Rightarrow \ L;\top} \ \textsc{typ\_ret}\mathrm{T}$$

$$\frac{}{H;G;L;\texttt{void} \vdash \ \texttt{return}; \ \Rightarrow \ L;\top} \ \textsc{typ\_ret}\mathrm{Void}$$

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

$$\frac{H\,;G\,;L_0\,;rt \vdash_{ss} stmt_1 \mathinner{..} stmt_n \Rightarrow L_n\,;r}{H\,;G\,;L_0\,;rt \vdash \{stmt_1 \mathinner{..} stmt_n\} \;\; ;r} \quad \text{TYP\_BLOCK}$$

$\boxed{H\,;G\,;L_0\,;rt \vdash_{ss} stmt_1 \mathinner{..} stmt_n \Rightarrow L_n\,;returns}$

$$\frac{\begin{array}{l} H;G;L_0;rt \vdash\ stmt_1 \ \Rightarrow\ L_1;\bot \\ \dots \\ H;G;L_{n-2};rt \vdash\ stmt_{n-1} \ \Rightarrow\ L_{n-1};\bot \\ H;G;L_{n-1};rt \vdash\ stmt_n \ \Rightarrow\ L_n;r \end{array}}{H\,;G\,;L_0\,;rt \vdash_{ss} stmt_1 \mathinner{..} stmt_{n-1}\, stmt_n \Rightarrow L_n\,;r} \quad \text{TYP\_STMTS}$$

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

$$\frac{H \vdash t_1 \quad .. \quad H \vdash t_i \quad x_1 \mathinner{..} x_i\ \textbf{distinct}}{H\,;G \vdash_s \texttt{struct}\ S\{\ t_1\ x_1\,;\ ..\ ;t_i\ x_i\ \}} \quad \text{TYP\_TDECLOK}$$

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

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

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

$$\frac{}{H\,;G \vdash \epsilon} \quad \text{TYP\_DEMPTY}$$

$$\frac{H\,;G \vdash prog}{H\,;G \vdash gdecl\, prog} \quad \text{TYP\_DGDECL}$$

$$\frac{H\,;G \vdash_f fdecl \quad H\,;G \vdash prog}{H\,;G \vdash fdecl\, prog} \quad \text{TYP\_DFDECL}$$

$$\frac{H\,;G \vdash_s tdecl \quad H\,;G \vdash prog}{H\,;G \vdash tdecl\, prog} \quad \text{TYP\_DTDECL}$$

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

$$\frac{}{H\,;G \vdash_g \epsilon \Rightarrow G} \quad \text{TYP\_GEMPTY}$$

$$\frac{H\,;G_1 \vdash_g prog \Rightarrow G_2}{H\,;G_1 \vdash_g tdecl\, prog \Rightarrow G_2} \quad \text{TYP\_GTDECL}$$

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

$$\frac{H\,;G_1 \vdash_g prog \Rightarrow G_2}{H\,;G_1 \vdash_g fdecl\, prog \Rightarrow G_2} \quad \text{TYP\_GFDECL}$$

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

$$\frac{H \vdash_{rt} rt \quad H \vdash t_1 \quad .. \quad H \vdash t_n}{H \vdash rt\, f\,(t_1\,x_1,\,..\,,t_n\,x_n)\ block \Rightarrow f : (t_1,\,..\,,t_n)\ \texttt{->}\ rt} \quad \text{TYP\_FTYP}$$

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

$$\frac{}{H ; G \vdash_f \epsilon \Rightarrow G} \quad \text{TYP\_FEMPTY}$$

$$\frac{H ; G_1 \vdash_f \mathit{prog} \Rightarrow G_2}{H ; G_1 \vdash_f \mathit{tdecl}\,\mathit{prog} \Rightarrow G_2} \quad \text{TYP\_FTDECL}$$

$$\frac{H ; G_1 \vdash_f \mathit{prog} \Rightarrow G_2}{H ; G_1 \vdash_f \mathit{gdecl}\,\mathit{prog} \Rightarrow G_2} \quad \text{TYP\_FGDECL}$$

$$\frac{\begin{array}{l} H \vdash \mathit{fdecl} \Rightarrow f : t \\ f \notin G_1 \quad H ; G_1, f : t \vdash_f \mathit{prog} \Rightarrow G_2 \end{array}}{H ; G_1 \vdash_f \mathit{fdecl}\,\mathit{prog} \Rightarrow G_2} \quad \text{TYP\_FFDECL}$$

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

$$\frac{}{H \vdash_s \epsilon \Rightarrow H} \quad \text{TYP\_SEMPTY}$$

$$\frac{S \notin H_1 \quad H_1, \texttt{struct}\ S\{\ \mathit{fields}\ \} \ \vdash_s \mathit{prog} \Rightarrow H_2}{H_1 \vdash_s \texttt{struct}\ S\{\ \mathit{fields}\ \} \ \ \mathit{prog} \Rightarrow H_2} \quad \text{TYP\_STDECL}$$

$$\frac{H_1 \vdash_s \mathit{prog} \Rightarrow H_2}{H_1 \vdash_s \mathit{gdecl}\,\mathit{prog} \Rightarrow H_2} \quad \text{TYP\_SGDECL}$$

$$\frac{H_1 \vdash_s \mathit{prog} \Rightarrow H_2}{H_1 \vdash_s \mathit{fdecl}\,\mathit{prog} \Rightarrow H_2} \quad \text{TYP\_SFDECL}$$

$\boxed{\vdash \mathit{prog}}$

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

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 $\bot$ 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.