C-Like Language with Objects (CLO)'s Type System and Specification

n	Constant int		
b	Constant bool		
$cstr$	Constant string		
id	Identifiers		
cid	Class identifiers		
$j,\ k,\ m$	Index		
typ	::=		Types
	|	`bot`	bottom
	|	`bool`	bool
	|	`int`	int
	|	ref	reference
	|	ref `?`	nullable
ref	::=		References
	|	`string`	string
	|	cid	class
	|	typ `[]`	array
$unop$	::=		Unary operators
	|	`-`	unary signed negation
	|	`!`	unary logical negation
	|	`~`	unary bitwise negation
$binop$	::=		Binary operators
	|	`+`	binary signed addition
	|	`*`	binary signed multiplication
	|	`-`	binary signed subtraction
	|	`==`	binary equality
	|	`!=`	binary inequality
	|	`<`	binary signed less-than
	|	`<=`	binary signed less-than or equals
	|	`>`	binary signed greater-than
	|	`>=`	binary signed greater-than or equals
	|	`&`	binary bool bitwise and
	|	`|`	binary bool bitwise or
	|	`[&]`	binary int bitwise and
	|	`[|]`	binary int bitwise or
	|	`<<`	binary shift left

	$\vert$	`>>`	binary logical shift right
	$\vert$	`>>>`	binary arithmetic shift right
$const$	$::=$		Constants
	$\vert$	`null`	null
	$\vert$	b	bool
	$\vert$	n	int
	$\vert$	$cstr$	string
$path$	$::=$		Paths
	$\vert$	$\texttt{this}\,.\,id$	identifiers in this class
	$\vert$	$lhs\_or\_call\,.\,id$	path identifiers, e.g, a.b.f().c
$call$	$::=$		Calls
	$\vert$	$id\;(\;\overline{exp_j}^{\,j}\;)$	global functions
	$\vert$	$\texttt{super}\,.\,id\;(\;\overline{exp_j}^{\,j}\;)$	super methods
	$\vert$	$path\;(\;\overline{exp_j}^{\,j}\;)$	path methods, e.g. a.f().b.g()
lhs_or_call	$::=$		Left-hand sides or calls
	$\vert$	lhs	left-hand sides
	$\vert$	$call$	calls
lhs	$::=$		Left-hand sides
	$\vert$	id	variables
	$\vert$	$path$	paths
	$\vert$	$lhs\_or\_call\,[\,exp\,]$	array index
exp	$::=$		Expressions
	$\vert$	$const$	constant
	$\vert$	`this`	this
	$\vert$	lhs_or_call	left-hand sides or calls
	$\vert$	$\texttt{new}\;typ\,[\,exp_1\,]\;(\,\texttt{fun}\;id\texttt{->}exp_2\,)$	new array creation
	$\vert$	$\texttt{new}\;cid\;(\;\overline{exp_j}^{\,j}\;)$	constructor
	$\vert$	$binop\;exp_1\;exp_2$	binary arithmetic
	$\vert$	$unop\;exp$	unary arithmetic
exp_{opt}	$::=$		Optional expressions
	$\vert$	ϵ	None
	$\vert$	exp	Some exp
$init$	$::=$		Initializer
	$\vert$	exp	expression
	$\vert$	$\{\;\overline{init_j}^{\,j\in 1..m}\;\}$	constant array
$vdecl$	$::=$		Variable declarations
	$\vert$	$typ\;id\texttt{=}init\texttt{;}$	
$vdecls$	$::=$		List of variable declarations
	$\vert$	ϵ	Empty

	$\mid$	*vdecl vdecls*	Cons
stmt	::=		Statements
	$\mid$	*lhs*=*exp*`;`	assignment
	$\mid$	*call*`;`	call
	$\mid$	`fail(`*exp*`);`	fail
	$\mid$	`if(`*exp*`)` *stmt* [`else` $stmt_{opt}$]	if-then, optional else
	$\mid$	`if?(`*ref id* = *exp*`)` *stmt* [`else` $stmt_{opt}$]	if null, optional else
	$\mid$	`cast(`*cid id* = *exp*`)` *stmt* [`else` $stmt_{opt}$]	checked cast, optional else
	$\mid$	`while(`*exp*`)` *stmt*	while loop
	$\mid$	`for(`*vdecls*`;` exp_{opt}`;` $stmt_{opt}$`)` *stmt*	for loop
	$\mid$	{*block*}	block
stmts	::=		List of statements
	$\mid$	ϵ	Empty
	$\mid$	*stmt stmts*	Cons
$stmt_{opt}$	::=		Optional statements
	$\mid$	ϵ	none
	$\mid$	*stmt*	some
block	::=		Blocks
	$\mid$	*vdecls stmts*	
args	::=		A list of arguments
	$\mid$	ϵ	
	$\mid$	*typ id*`,` *args*	
rtyp	::=		Return types
	$\mid$	*typ*	function - return type
	$\mid$	`unit`	procedure - no return value
efdecl	::=		External function declarations
	$\mid$	*rtyp id*`(`*args*`)` `extern`	
fdecl	::=		Function declarations
	$\mid$	*typ id*`(`*args*`)`{*block* `return` *exp*`;`}	function
	$\mid$	`unit` *id*`(`*args*`)`{*block* `return;`}	procedure
fdecls	::=		A list of method declarations
	$\mid$	ϵ	nil
	$\mid$	*fdecl fdecls*	cons
cinits	::=		A list of field initialization
	$\mid$	ϵ	nil
	$\mid$	`this.`*id*=*init*`;` *cinits*	cons
ctor	::=		Constructors
	$\mid$	`new(`*args*`)(` $\overline{exp_j}^{\,j}$ `)` *cinits*{*block*}	

cid_{ext}	$::=$		Optional extensions
	$\vert$	ϵ	implicitly extend `Object`
	$\vert$	$\texttt{<:}\,cid$	extend cid
$fields$	$::=$		A list of field declarations
	$\vert$	ϵ	nil
	$\vert$	$typ\ id\,\texttt{;}\,fields$	cons
$cdecl$	$::=$		Classes
	$\vert$	$\texttt{class}\ cid\ cid_{ext}\{fields\ ctor\ fdecls\}\texttt{;}$	
$gdecl$	$::=$		Global declarations
	$\vert$	$vdecl$	global variables (constants)
	$\vert$	$fdecl$	function declarations
	$\vert$	$efdecl$	external function declarations
	$\vert$	$cdecl$	class declarations
$prog$	$::=$		Programs
	$\vert$	ϵ	
	$\vert$	$gdecl\ prog$	
Γ	$::=$		Local variable contexts
	$\vert$	$\cdot$	empty
	$\vert$	$\Gamma\texttt{,}\ id\,\texttt{:}\,typ$	extended with local id of type typ
$ftyp$	$::=$		Function types
	$\vert$	$\texttt{(}\ \overline{typ_j}^{\,j}\ \texttt{)}\ \texttt{->}\,rtyp$	
$gtyp$	$::=$		Global types
	$\vert$	$ftyp$	functions
	$\vert$	typ	values
Δ	$::=$		Global identifiers and function contexts
	$\vert$	$\cdot$	empty
	$\vert$	$\Delta, id: gtyp$	extended with global value or function type
Θ	$::=$		Method contexts
	$\vert$	$\cdot$	empty
	$\vert$	$\Theta, id: ftyp$	cons
Φ	$::=$		Field contexts
	$\vert$	$\cdot$	empty
	$\vert$	$\Phi, id: typ$	cons
Σ	$::=$		Class signatures
	$\vert$	$\cdot$	empty
	$\vert$	$\Sigma\texttt{,}\ cid\ cid_{ext}\{\Phi\texttt{;}\ \overline{typ_j}^{\,j}\ \texttt{;}\,\Theta\}$	cons
cid_{opt}	$::=$		Optional class
	$\vert$	cid	some in the scope of class cid

	$\mid$	$-$	none, otherwise
typ_{opt}	$::=$		Optional typ, the return of getField
	$\mid$	$\texttt{Some}\ typ$	some
	$\mid$	$\texttt{None}$	none
$ftyp_{opt}$	$::=$		Optional ftyp, the return of getMethod
	$\mid$	$\texttt{Some}\ ftyp$	some
	$\mid$	$\texttt{None}$	none
$Ctxt$	$::=$		Contexts
	$\mid$	$\Sigma\texttt{;}\Delta\texttt{;}\Gamma\texttt{;}\ cid_{opt}$	Class decls, globals, locals, containing class

$\boxed{\Sigma \vdash typ}$ Σ shows that typ is well-formed.

$$\frac{}{\Sigma \vdash \texttt{bot}} \quad \text{TYP\_BOT}$$

$$\frac{}{\Sigma \vdash \texttt{bool}} \quad \text{TYP\_BOOL}$$

$$\frac{}{\Sigma \vdash \texttt{int}} \quad \text{TYP\_INT}$$

$$\frac{\Sigma \vdash_r ref}{\Sigma \vdash ref} \quad \text{TYP\_REF}$$

$$\frac{\Sigma \vdash_r ref}{\Sigma \vdash ref\,\texttt{?}} \quad \text{TYP\_NULLABLE}$$

$\boxed{\Sigma \vdash_r ref}$ Σ shows that ref is well-formed.

$$\frac{}{\Sigma \vdash_r \texttt{string}} \quad \text{REF\_STRING}$$

$$\frac{cid\ cid_{ext}\{\Phi\texttt{;}\ \overline{typ_j}^{\,j}\ \texttt{;}\Theta\} \in \Sigma}{\Sigma \vdash_r cid} \quad \text{REF\_CLASS}$$

$$\frac{\Sigma \vdash typ}{\Sigma \vdash_r typ\,\texttt{[]}} \quad \text{REF\_ARRAY}$$

$\boxed{\Sigma \vdash typ_1 \texttt{<:}\ typ_2}$ Σ shows that typ_1 is a subtype of typ_2.

$$\frac{}{\Sigma \vdash \texttt{bool} <: \texttt{bool}} \quad \text{ST\_BOOL}$$

$$\frac{}{\Sigma \vdash \texttt{int} <: \texttt{int}} \quad \text{ST\_INT}$$

$$\frac{\Sigma \vdash_r \mathit{ref}_1 <: \mathit{ref}_2}{\Sigma \vdash \mathit{ref}_1 <: \mathit{ref}_2} \quad \text{ST\_REF}$$

$$\frac{\Sigma \vdash_r \mathit{ref}_1 <: \mathit{ref}_2}{\Sigma \vdash \mathit{ref}_1\,? <: \mathit{ref}_2\,?} \quad \text{ST\_NULLABLE}$$

$$\frac{\Sigma \vdash_r \mathit{ref}_1 <: \mathit{ref}_2}{\Sigma \vdash \mathit{ref}_1 <: \mathit{ref}_2\,?} \quad \text{ST\_REF\_NULLABLE}$$

$$\frac{}{\Sigma \vdash \texttt{bot} <: \mathit{ref}\,?} \quad \text{ST\_NULL\_NULLABLE}$$

$\boxed{\Sigma \vdash_r \mathit{ref}_1 <: \mathit{ref}_2}$ Σ shows that ref_1 is a sub-reference of ref_2.

$$\frac{}{\Sigma \vdash_r \texttt{string} <: \texttt{string}} \quad \text{SR\_STRING}$$

$$\frac{}{\Sigma \vdash_r \mathit{typ}\,\texttt{[]} <: \mathit{typ}\,\texttt{[]}} \quad \text{SR\_ARRAY}$$

$$\frac{\Sigma \vdash_c \mathit{cid}_1 <: \mathit{cid}_2}{\Sigma \vdash_r \mathit{cid}_1 <: \mathit{cid}_2} \quad \text{SR\_CLASS}$$

$\boxed{\Sigma \vdash_c \mathit{cid}_1 <: \mathit{cid}_2}$ Σ shows that cid_1 is a sub-class of cid_2.

$$\frac{\mathit{cid}\ \mathit{cid}_{ext}\{\Phi;\, \overline{\mathit{typ}_j}^{\,j}\,;\Theta\} \in \Sigma}{\Sigma \vdash_c \mathit{cid} <: \mathit{cid}} \quad \text{SC\_REFL}$$

$$\frac{\mathit{cid} <: \mathit{cid}_2\{\Phi;\, \overline{\mathit{typ}_j}^{\,j}\,;\Theta\} \in \Sigma \quad \Sigma \vdash_c \mathit{cid}_2 <: \mathit{cid}_3}{\Sigma \vdash_c \mathit{cid}_1 <: \mathit{cid}_3} \quad \text{SC\_TRANS}$$

$\boxed{\texttt{get\_field}\,\Sigma\ \mathit{cid}.\mathit{id} \ = \ \mathit{typ}_{opt}}$ Look up the type of field id in class cid.

$$\frac{cid\ cid_{ext}\{\Phi;\ \overline{typ_j}^{\,j}\ ;\Theta\}\ \in\ \Sigma \quad id : typ\ \in\ \Phi}{\texttt{get\_field}\ \Sigma\ cid.id\ \ =\ \ \texttt{Some}\ typ}\ \textsc{getfield\_base\_some}$$

$$\frac{cid\ \epsilon\{\Phi;\ \overline{typ_j}^{\,j}\ ;\Theta\}\ \in\ \Sigma \quad id\ \notin\ \Phi}{\texttt{get\_field}\ \Sigma\ cid.id\ \ =\ \ \texttt{None}}\ \textsc{getfield\_base\_none}$$

$$\frac{cid_1 <: cid_2\{\Phi;\ \overline{typ_j}^{\,j}\ ;\Theta\}\ \in\ \Sigma \quad id\ \notin\ \Phi \quad \texttt{get\_field}\ \Sigma\ cid_2.id\ \ =\ \ typ_{opt}}{\texttt{get\_field}\ \Sigma\ cid_1.id\ \ =\ \ typ_{opt}}\ \textsc{getfield\_inheritance}$$

$\boxed{\texttt{get\_method}\ \Sigma\ cid.id\ \ =\ \ ftyp_{opt}}$ Look up the type of method id in class cid.

$$\frac{cid\ cid_{ext}\{\Phi;\ \overline{typ_j}^{\,j}\ ;\Theta\}\ \in\ \Sigma \quad id : ftyp\ \in\ \Theta}{\texttt{get\_method}\ \Sigma\ cid.id\ \ =\ \ \texttt{Some}\ ftyp}\ \textsc{getmethod\_base\_some}$$

$$\frac{cid\ \epsilon\{\Phi;\ \overline{typ_j}^{\,j}\ ;\Theta\}\ \in\ \Sigma \quad id\ \notin\ \Theta}{\texttt{get\_method}\ \Sigma\ cid.id\ \ =\ \ \texttt{None}}\ \textsc{getmethod\_base\_none}$$

$$\frac{cid_1 <: cid_2\{\Phi;\ \overline{typ_j}^{\,j}\ ;\Theta\}\ \in\ \Sigma \quad id\ \notin\ \Theta \quad \texttt{get\_method}\ \Sigma\ cid_2.id\ \ =\ \ ftyp_{opt}}{\texttt{get\_method}\ \Sigma\ cid_1.id\ \ =\ \ ftyp_{opt}}\ \textsc{getmethod\_inheritance}$$

$\boxed{\vdash const : typ}$ $const$ has type typ.

$$\frac{}{\vdash \texttt{null} : \texttt{bot}}\ \textsc{const\_bot}$$

$$\frac{}{\vdash b : \texttt{bool}}\ \textsc{const\_bool}$$

$$\frac{}{\vdash n : \texttt{int}}\ \textsc{const\_int}$$

$$\frac{}{\vdash cstr : \texttt{string}}\ \textsc{const\_string}$$

$\boxed{\vdash binop\ \ : ftyp}$ $binop$ is of type $ftyp$.

$$\frac{}{\vdash \texttt{+ : (int,int)->int}}\ \textsc{bintyp\_plus}$$

$$\frac{}{\vdash \star : \texttt{(int,int)->int}}\ \text{BINTYP\_TIMES}$$

$$\frac{}{\vdash - : \texttt{(int,int)->int}}\ \text{BINTYP\_MINUS}$$

$$\frac{}{\vdash \texttt{==} : \texttt{(}typ\texttt{,}\,typ\texttt{)->bool}}\ \text{BINTYP\_EQ}$$

$$\frac{}{\vdash \texttt{!=} : \texttt{(}typ\texttt{,}\,typ\texttt{)->bool}}\ \text{BINTYP\_NEQ}$$

$$\frac{}{\vdash < : \texttt{(int,int)->bool}}\ \text{BINTYP\_LT}$$

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

$$\frac{}{\vdash > : \texttt{(int,int)->bool}}\ \text{BINTYP\_GE}$$

$$\frac{}{\vdash \texttt{>=} : \texttt{(int,int)->bool}}\ \text{BINTYP\_GTE}$$

$$\frac{}{\vdash \texttt{[\&]} : \texttt{(int,int)->int}}\ \text{BINTYP\_IAND}$$

$$\frac{}{\vdash \texttt{\&} : \texttt{(bool,bool)->bool}}\ \text{BINTYP\_AND}$$

$$\frac{}{\vdash \texttt{[|]} : \texttt{(int,int)->int}}\ \text{BINTYP\_IOR}$$

$$\frac{}{\vdash \texttt{|} : \texttt{(bool,bool)->bool}}\ \text{BINTYP\_OR}$$

$$\frac{}{\vdash \texttt{<<} : \texttt{(int,int)->int}}\ \text{BINTYP\_SHL}$$

$$\frac{}{\vdash \texttt{>>} : \texttt{(int,int)->int}}\ \text{BINTYP\_SHR}$$

$$\frac{}{\vdash \texttt{>>>} : \texttt{(int,int)->int}} \quad \text{BINTYP\_SAR}$$

$\boxed{\vdash unop : ftyp}$ $unop$ is of type $ftyp$.

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

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

$$\frac{}{\vdash \sim : \texttt{(int)->int}} \quad \text{UTYP\_NOT}$$

$\boxed{Ctxt \vdash_p path : gtyp}$ Show that $path$ has type $gtyp$.

$$\frac{\texttt{get\_field}\,\Sigma\; cid.id \;=\; \texttt{Some}\; typ \quad \texttt{get\_method}\,\Sigma\; cid.id \;=\; \texttt{None}}{\Sigma;\Delta;\Gamma;cid \vdash_p \texttt{this}\,.\,id : typ} \quad \text{PATH\_THIS\_FIELD}$$

$$\frac{\texttt{get\_method}\,\Sigma\; cid.id \;=\; \texttt{Some}\; ftyp \quad \texttt{get\_field}\,\Sigma\; cid.id \;=\; \texttt{None}}{\Sigma;\Delta;\Gamma;cid \vdash_p \texttt{this}\,.\,id : ftyp} \quad \text{PATH\_THIS\_METHOD}$$

$$\frac{\begin{array}{l} Ctxt \vdash_l lhs\_or\_call : cid \\ \texttt{get\_field}\,\Sigma\; cid.id \;=\; \texttt{Some}\; typ \quad \texttt{get\_method}\,\Sigma\; cid.id \;=\; \texttt{None}\end{array}}{Ctxt \vdash_p lhs\_or\_call\,.\,id : typ} \quad \text{PATH\_PATH\_FIELD}$$

$$\frac{\begin{array}{l} Ctxt \vdash_l lhs\_or\_call : cid \\ \texttt{get\_method}\,\Sigma\; cid.id \;=\; \texttt{Some}\; ftyp \quad \texttt{get\_field}\,\Sigma\; cid.id \;=\; \texttt{None}\end{array}}{Ctxt \vdash_p lhs\_or\_call\,.\,id : ftyp} \quad \text{PATH\_PATH\_METHOD}$$

$\boxed{Ctxt \vdash call : rtyp}$ Show that $call$ has type $rtyp$.

$$\frac{id : \texttt{(}\; \overline{typ_j}^{\,j}\; \texttt{)->} rtyp \;\in\; \Delta \quad \overline{Ctxt \vdash exp_j \;\texttt{<:}\; typ_j}^{\,j}}{Ctxt \vdash id\; \texttt{(}\; \overline{exp_j}^{\,j}\; \texttt{)} : rtyp} \quad \text{CALL\_FUNC}$$

$$\frac{id \;\notin\; \Delta \quad id : \texttt{(}\; \overline{typ_j}^{\,j}\; \texttt{)->} rtyp \quad \overline{Ctxt \vdash exp_j \;\texttt{<:}\; typ_j}^{\,j}}{Ctxt \vdash id\; \texttt{(}\; \overline{exp_j}^{\,j}\; \texttt{)} : rtyp} \quad \text{CALL\_BUILTIN}$$

$$\frac{\begin{array}{l} cid_1 \mathrel{<:} cid_2 \{\Phi ; \overline{typ'_k}^{\,k} ; \Theta\} \in \Sigma \quad \texttt{get\_method}\ \Sigma\ cid_2.id \ = \ \texttt{Some}\ (\ \overline{typ_j}^{\,j}\ ) \texttt{->} rtyp \\ \overline{Ctxt \vdash exp_j \mathrel{<:} typ_j}^{\,j} \end{array}}{\Sigma ; \Delta ; \Gamma ; cid_1 \vdash \texttt{super}\ .\ id\ (\ \overline{exp_j}^{\,j}\ ) : rtyp} \quad \text{CALL\_SUPER\_METHOD}$$

$$\frac{Ctxt \vdash_p path : (\ \overline{typ_j}^{\,j}\ ) \texttt{->} rtyp \quad \overline{Ctxt \vdash exp_j \mathrel{<:} typ_j}^{\,j}}{Ctxt \vdash path\ (\ \overline{exp_j}^{\,j}\ ) : rtyp} \quad \text{CALL\_PATH\_METHOD}$$

$\boxed{Ctxt \vdash_l lhs\_or\_call : typ}$ Show that lhs_or_call has type typ.

$$\frac{Ctxt \vdash lhs : typ}{Ctxt \vdash_l lhs : typ} \quad \text{LC\_LHS}$$

$$\frac{Ctxt \vdash call : typ}{Ctxt \vdash_l call : typ} \quad \text{LC\_CALL}$$

$\boxed{Ctxt \vdash_l lhs : typ}$ Show that lhs has type typ.

$$\frac{id : typ \in \Gamma}{Ctxt \vdash_l id : typ} \quad \text{LHS\_LOCAL\_VAR}$$

$$\frac{id \notin \Gamma \quad id : typ \in \Delta}{Ctxt \vdash_l id : typ} \quad \text{LHS\_GLOBAL\_VAR}$$

$$\frac{Ctxt \vdash_l lhs : typ\,[\,] \quad Ctxt \vdash exp : \texttt{int}}{Ctxt \vdash_l lhs\,[\,exp\,] : typ} \quad \text{LHS\_INDEX}$$

$\boxed{Ctxt \vdash exp : typ}$ Show that exp has type typ.

$$\frac{\vdash const : typ}{Ctxt \vdash const : typ} \quad \text{EXP\_CONST}$$

$$\frac{}{\Sigma ; \Delta ; \Gamma ; cid \vdash \texttt{this} : cid} \quad \text{EXP\_THIS}$$

$$\frac{Ctxt \vdash exp_1 : \texttt{int} \quad \Sigma ; \Delta ; \Gamma , id : \texttt{int} ; cid_{opt} \vdash exp_2 \mathrel{<:} typ}{Ctxt \vdash \texttt{new}\ typ\,[\,exp_1\,]\ (\ \texttt{fun}\ id \texttt{->} exp_2\,) : typ\,[\,]} \quad \text{EXP\_NEW}$$

$$\frac{cid\ cid_{ext}\{\Phi;\ \overline{typ_j}^{\,j}\ ;\Theta\} \in \Sigma \quad \overline{Ctxt \vdash exp_j <:\ typ_j}^{\,j}}{Ctxt \vdash \texttt{new}\ cid\ (\ \overline{exp_j}^{\,j}\ )\ \texttt{:}\ cid} \quad \text{EXP\_CTOR}$$

$$\frac{Ctxt \vdash exp_1 \texttt{:} typ_1 \quad Ctxt \vdash exp_2 \texttt{:} typ_2 \quad \vdash binop\ \texttt{:}\ \texttt{(}typ_1\texttt{,}\ typ_2\texttt{)}\texttt{->}typ}{Ctxt \vdash binop\ exp_1\ exp_2 \texttt{:} typ} \quad \text{EXP\_BINOP}$$

$$\frac{Ctxt \vdash exp \texttt{:} typ \quad \vdash unop\ \texttt{:}\ \texttt{(}typ\texttt{)}\texttt{->}typ'}{Ctxt \vdash unop\ exp \texttt{:} typ'} \quad \text{EXP\_UNOP}$$

$$\frac{Ctxt \vdash_l lhs\_or\_call \texttt{:} typ}{Ctxt \vdash lhs\_or\_call \texttt{:} typ} \quad \text{EXP\_LHS\_OR\_CALL}$$

$$\frac{Ctxt \vdash exp \texttt{:} typ\,\texttt{[]}}{Ctxt \vdash \texttt{length\_of\_array}\ \texttt{(}exp\texttt{)}\texttt{:int}} \quad \text{EXP\_LENGTH\_OF\_ARRAY}$$

$\boxed{Ctxt \vdash_{opt} exp_{opt} \texttt{:}\ \texttt{bool}}$ Show that exp_{opt} has type `bool` .

$$\frac{}{Ctxt \vdash_{opt} \epsilon \texttt{:}\ \texttt{bool}} \quad \text{OPT\_EXP\_NONE}$$

$$\frac{Ctxt \vdash exp \texttt{:bool}}{Ctxt \vdash_{opt} exp \texttt{:}\ \texttt{bool}} \quad \text{OPT\_EXP\_SOME}$$

$\boxed{Ctxt \vdash exp <:\ typ}$ Show that exp has a subtype of typ.

$$\frac{Ctxt \vdash exp \texttt{:} typ' \quad \Sigma \vdash typ' <:\ typ}{Ctxt \vdash exp <:\ typ} \quad \text{EXPSUB\_INTRO}$$

$\boxed{Ctxt\texttt{;}\ typ \vdash_i init\ \texttt{ok}}$ Show that $init$ has a subtype of expected type typ.

$$\frac{Ctxt \vdash exp <:\ typ}{Ctxt\texttt{;}\ typ \vdash_i exp\ \texttt{ok}} \quad \text{INIT\_EXP}$$

$$\frac{\overline{Ctxt\texttt{;}\ typ \vdash_i init_j\ \texttt{ok}}^{\,j\in 1..m}}{Ctxt\texttt{;}\ typ\,\texttt{[]} \vdash_i \{\ \overline{init_j}^{\,j\in 1..m}\ \}\ \texttt{ok}} \quad \text{INIT\_ARRAY}$$

$\boxed{Ctxt \vdash vdecls \Rightarrow Ctxt'}$ $vdecls$ are well-typed; extend the local context to be $Ctxt'$

$$\frac{}{\mathit{Ctxt} \vdash \epsilon \Rightarrow \mathit{Ctxt}} \quad \text{VDECLS\_NIL}$$

$$\frac{\mathit{id} \notin \Gamma \quad \mathit{Ctxt}\,\mathtt{;}\,\mathit{typ} \vdash_i \mathit{init}\ \mathtt{ok} \quad \Sigma\mathtt{;}\Delta\mathtt{;}\Gamma\mathtt{,}\,\mathit{id}\,\mathtt{:}\,\mathit{typ}\,\mathtt{;}\,\mathit{cid}_{opt} \vdash \mathit{vdecls} \Rightarrow \Sigma\mathtt{;}\Delta\mathtt{;}\Gamma'\mathtt{;}\,\mathit{cid}_{opt}}{\mathit{Ctxt} \vdash \mathit{typ}\ \mathit{id}\mathtt{=}\mathit{init}\mathtt{;}\ \mathit{vdecls} \Rightarrow \Sigma\mathtt{;}\Delta\mathtt{;}\Gamma'\mathtt{;}\,\mathit{cid}_{opt}} \quad \text{VDECLS\_CONS}$$

$\boxed{\mathit{Ctxt} \vdash \mathit{stmt}\ \mathtt{ok}}$ Typecheck a statement in the given context.

$$\frac{\mathit{Ctxt} \vdash_l \mathit{lhs}\,\mathtt{:}\,\mathit{typ} \quad \mathit{Ctxt} \vdash \mathit{exp}\ \mathtt{<:}\ \mathit{typ}}{\mathit{Ctxt} \vdash \mathit{lhs}\mathtt{=}\mathit{exp}\mathtt{;}\ \mathtt{ok}} \quad \text{STMT\_ASSIGN}$$

$$\frac{\mathit{Ctxt} \vdash \mathit{call}\,\mathtt{:unit}}{\mathit{Ctxt} \vdash \mathit{call}\mathtt{;}\ \mathtt{ok}} \quad \text{STMT\_CALL}$$

$$\frac{\mathit{Ctxt} \vdash \mathit{exp}\,\mathtt{:string}}{\mathit{Ctxt} \vdash \mathtt{fail}\ \mathtt{(}\mathit{exp}\mathtt{);}\ \mathtt{ok}} \quad \text{STMT\_FAIL}$$

$$\frac{\begin{array}{l}\mathit{Ctxt} \vdash \mathit{exp}\,\mathtt{:bool} \quad \mathit{Ctxt} \vdash \mathit{stmt}\ \mathtt{ok}\\ [\mathit{Ctxt} \vdash_{opt} \mathit{stmt}_{opt}\ \mathtt{ok}]\end{array}}{\mathit{Ctxt} \vdash \mathtt{if}\,\mathtt{(}\mathit{exp}\mathtt{)}\ \mathit{stmt}\ [\mathtt{else}\ \mathit{stmt}_{opt}]\ \mathtt{ok}} \quad \text{STMT\_IF}$$

$$\frac{\begin{array}{l}\mathit{Ctxt} \vdash \mathit{exp}\ \mathtt{<:}\ \mathit{ref}\mathtt{?} \quad \Sigma\mathtt{;}\Delta\mathtt{;}\Gamma\mathtt{,}\,\mathit{id}\,\mathtt{:}\,\mathit{ref}\,\mathtt{;}\,\mathit{cid}_{opt} \vdash \mathit{stmt}\ \mathtt{ok}\\ [\mathit{Ctxt} \vdash_{opt} \mathit{stmt}_{opt}\ \mathtt{ok}]\end{array}}{\mathit{Ctxt} \vdash \mathtt{if?(}\mathit{ref}\ \mathit{id}\ \mathtt{=}\ \mathit{exp}\mathtt{)}\ \mathit{stmt}\ [\mathtt{else}\ \mathit{stmt}_{opt}]\ \mathtt{ok}} \quad \text{STMT\_IFNULL}$$

$$\frac{\begin{array}{l}\mathit{Ctxt} \vdash \mathit{exp}\ \mathtt{<:}\ \mathit{cid}' \quad \Sigma \vdash_c \mathit{cid}\ \mathtt{<:}\ \mathit{cid}'\\ \Sigma\mathtt{;}\Delta\mathtt{;}\Gamma\mathtt{,}\,\mathit{id}\,\mathtt{:}\,\mathit{cid}\,\mathtt{;}\,\mathit{cid}_{opt} \vdash \mathit{stmt}\ \mathtt{ok}\\ [\mathit{Ctxt} \vdash_{opt} \mathit{stmt}_{opt}\ \mathtt{ok}]\end{array}}{\mathit{Ctxt} \vdash \mathtt{cast}\,\mathtt{(}\mathit{cid}\ \mathit{id}\ \mathtt{=}\ \mathit{exp}\mathtt{)}\ \mathit{stmt}\ [\mathtt{else}\ \mathit{stmt}_{opt}]\ \mathtt{ok}} \quad \text{STMT\_CAST}$$

$$\frac{\begin{array}{l}\mathit{Ctxt} \vdash \mathit{vdecls} \Rightarrow \Sigma\mathtt{;}\Delta\mathtt{;}\Gamma'\mathtt{;}\,\mathit{cid}_{opt} \quad \Sigma\mathtt{;}\Delta\mathtt{;}\Gamma'\mathtt{;}\,\mathit{cid}_{opt} \vdash_{opt} \mathit{exp}_{opt}\,\mathtt{:bool}\\ [\Sigma\mathtt{;}\Delta\mathtt{;}\Gamma'\mathtt{;}\,\mathit{cid}_{opt} \vdash_{opt} \mathit{stmt}_{opt}\ \mathtt{ok}] \quad \Sigma\mathtt{;}\Delta\mathtt{;}\Gamma'\mathtt{;}\,\mathit{cid}_{opt} \vdash \mathit{stmt}\ \mathtt{ok}\end{array}}{\mathit{Ctxt} \vdash \mathtt{for}\ \mathtt{(}\mathit{vdecls}\mathtt{;}\,\mathit{exp}_{opt}\mathtt{;}\,\mathit{stmt}_{opt}\mathtt{)}\,\mathit{stmt}\ \mathtt{ok}} \quad \text{STMT\_FOR}$$

$$\frac{\mathit{Ctxt} \vdash \mathit{exp}\,\mathtt{:bool} \quad \mathit{Ctxt} \vdash \mathit{stmt}\ \mathtt{ok}}{\mathit{Ctxt} \vdash \mathtt{while}\ \mathtt{(}\mathit{exp}\mathtt{)}\,\mathit{stmt}\ \mathtt{ok}} \quad \text{STMT\_WHILE}$$

$$\frac{\mathit{Ctxt} \vdash \mathit{block} \Rightarrow \Sigma\mathtt{;}\Delta\mathtt{;}\Gamma'\mathtt{;}\,\mathit{cid}_{opt}}{\mathit{Ctxt} \vdash \{\mathit{block}\}\quad \mathtt{ok}} \quad \text{STMT\_BLOCK}$$

$\boxed{Ctxt \vdash stmts\ \texttt{ok}}$ Typecheck a sequence of statements.

$$\frac{}{Ctxt \vdash \epsilon\ \texttt{ok}}\quad \text{STMTS\_NIL}$$

$$\frac{Ctxt \vdash stmt\ \texttt{ok} \quad Ctxt \vdash stmts\ \texttt{ok}}{Ctxt \vdash stmt\ stmts\ \texttt{ok}}\quad \text{STMTS\_CONS}$$

$\boxed{[Ctxt \vdash_{opt} stmt_{opt}\ \texttt{ok}]}$ Typecheck an optional statement.

$$\frac{}{[Ctxt \vdash_{opt} \epsilon\ \texttt{ok}]}\quad \text{OPT\_STMT\_NONE}$$

$$\frac{Ctxt \vdash stmt\ \texttt{ok}}{[Ctxt \vdash_{opt} stmt\ \texttt{ok}]}\quad \text{OPT\_STMT\_SOME}$$

$\boxed{Ctxt \vdash block \Rightarrow Ctxt'}$ Typecheck a block, extending the local context to `G'` .

$$\frac{Ctxt \vdash vdecls \Rightarrow \Sigma\texttt{;}\Delta\texttt{;}\Gamma'\texttt{;}cid_{opt} \quad \Sigma\texttt{;}\Delta\texttt{;}\Gamma'\texttt{;}cid_{opt} \vdash stmts\ \texttt{ok}}{Ctxt \vdash vdecls\ stmts \Rightarrow \Sigma\texttt{;}\Delta\texttt{;}\Gamma'\texttt{;}cid_{opt}}\quad \text{BLOCK}$$

$\boxed{Ctxt \vdash args \Rightarrow Ctxt'}$ Show that *args* are well-scoped; extend the local variable context to be `G'` .

$$\frac{}{Ctxt \vdash \epsilon \Rightarrow Ctxt}\quad \text{ARGS\_NIL}$$

$$\frac{id \notin \Gamma \quad \Sigma\texttt{;}\Delta\texttt{;}\Gamma\texttt{,}id\texttt{:}typ\texttt{;}cid_{opt} \vdash args \Rightarrow \Sigma\texttt{;}\Delta\texttt{;}\Gamma'\texttt{;}cid_{opt}}{Ctxt \vdash typ\ id\texttt{,}\ args \Rightarrow \Sigma\texttt{;}\Delta\texttt{;}\Gamma'\texttt{;}cid_{opt}}\quad \text{ARGS\_CONS}$$

$\boxed{\Sigma\texttt{;}\Delta\texttt{;}\ \cdot\ \texttt{;}cid_{opt} \vdash fdecl\ \texttt{ok}}$ Typecheck a method, function or procedure body.

$$\frac{\begin{array}{l}\Sigma\texttt{;}\Delta\texttt{;}\ \cdot\ \texttt{;}cid_{opt} \vdash args \Rightarrow Ctxt \quad Ctxt \vdash block \Rightarrow \Sigma\texttt{;}\Delta\texttt{;}\Gamma'\texttt{;}cid_{opt}\\ \Sigma\texttt{;}\Delta\texttt{;}\Gamma'\texttt{;}cid_{opt} \vdash exp\ \texttt{<:}\ typ\end{array}}{\Sigma\texttt{;}\Delta\texttt{;}\ \cdot\ \texttt{;}cid_{opt} \vdash typ\ id\ \texttt{(}args\texttt{)}\ \{block\ \texttt{return}\ exp\texttt{;}\}\ \texttt{ok}}\quad \text{FDECL\_FUNC}$$

$$\frac{\Sigma\texttt{;}\Delta\texttt{;}\ \cdot\ \texttt{;}cid_{opt} \vdash args \Rightarrow Ctxt \quad Ctxt \vdash block \Rightarrow \Sigma\texttt{;}\Delta\texttt{;}\Gamma'\texttt{;}cid_{opt}}{\Sigma\texttt{;}\Delta\texttt{;}\ \cdot\ \texttt{;}cid_{opt} \vdash \texttt{unit}\ id\ \texttt{(}args\texttt{)}\ \{block\ \texttt{return}\ \texttt{;}\}\ \texttt{ok}}\quad \text{FDECL\_PROC}$$

$\boxed{\Sigma; \Delta; \cdot; cid \vdash fdecls\ \texttt{ok}}$ Typecheck a list of methods.

$$\frac{}{\Sigma; \Delta; \cdot; cid \vdash \epsilon \; \texttt{ok}} \quad \text{FDECLS\_NIL}$$

$$\frac{\Sigma\texttt{;}\,\Delta\texttt{;}\ \cdot\texttt{;}\ cid \vdash fdecl\ \texttt{ok} \quad \Sigma; \Delta; \cdot; cid \vdash fdecls\ \texttt{ok}}{\Sigma; \Delta; \cdot; cid \vdash fdecl\, fdecls\ \texttt{ok}} \quad \text{FDECLS\_CONS}$$

$\boxed{Ctxt \vdash id : ftyp \ \text{ can override } \ cid_{ext}}$ Show that id with type $ftyp$ can override parent class cid_{ext}.

$$\frac{}{Ctxt \vdash id : ftyp \ \text{ can override } \ \epsilon} \quad \text{OR\_OBJECT}$$

$$\frac{\texttt{get\_method}\,\Sigma\ cid.id \ = \ \texttt{None}}{Ctxt \vdash id : ftyp \ \text{ can override } \ \texttt{<:}\,cid} \quad \text{OR\_NOMETHOD}$$

$$\frac{\texttt{get\_method}\,\Sigma\ cid.id \ = \ \texttt{Some (}\,\overline{typ'_j}^{\,j}\,\texttt{)->}typ' \quad \overline{\Sigma \vdash typ'_j \ \texttt{<:}\ typ_j}^{\,j} \quad \Sigma \vdash typ \ \texttt{<:}\ typ'}{Ctxt \vdash id : \texttt{(}\,\overline{typ_j}^{\,j}\,\texttt{)->}typ \ \text{ can override } \ \texttt{<:}\,cid} \quad \text{OR\_FUNC}$$

$$\frac{\texttt{get\_method}\,\Sigma\ cid.id \ = \ \texttt{Some (}\,\overline{typ'_j}^{\,j}\,\texttt{)->unit} \quad \overline{\Sigma \vdash typ'_j \ \texttt{<:}\ typ_j}^{\,j}}{Ctxt \vdash id : \texttt{(}\,\overline{typ_j}^{\,j}\,\texttt{)->unit} \ \text{ can override } \ \texttt{<:}\,cid} \quad \text{OR\_PROC}$$

$\boxed{cid_{ext}; \Phi \vdash fields \Rightarrow \Phi'}$ Extend Φ to Φ' by adding field declarations (with parent class cid_{ext}).

$$\frac{}{cid_{ext}; \Phi \vdash \epsilon \Rightarrow \Phi} \quad \text{GENF\_NIL}$$

$$\frac{id \notin \Phi \quad \epsilon; \Phi, id\!: typ \vdash fields \Rightarrow \Phi'}{\epsilon; \Phi \vdash typ\ id\texttt{;}\,fields \Rightarrow \Phi'} \quad \text{GENF\_BASE}$$

$$\frac{id \notin \Phi \quad \texttt{get\_field}\,\Sigma\ cid.id \ = \ \texttt{None} \quad \texttt{<:}\,cid; \Phi, id\!: typ \vdash fields \Rightarrow \Phi'}{\texttt{<:}\,cid; \Phi \vdash typ\ id\texttt{;}\,fields \Rightarrow \Phi'} \quad \text{GENF\_INHERITANCE}$$

$\boxed{\Sigma; cid_{ext}; \Phi; \Theta \vdash fdecls \Rightarrow \Theta'}$ Extend Θ to Θ' by adding method declaratons (with parent class cid_{ext}).

$$\frac{}{\Sigma; cid_{ext}; \Phi; \Theta \vdash \epsilon \Rightarrow \Theta} \quad \text{GENM\_NIL}$$

$$\frac{id \notin \Phi\,\mathsf{and}\,\Theta \quad \Sigma; cid_{ext}; \Phi; \Theta, id\!: \texttt{(}\,\overline{typ_j}^{\,j}\,\texttt{)->}typ \vdash fdecls \Rightarrow \Theta'}{\Sigma; cid_{ext}; \Phi; \Theta \vdash typ\ id\ \texttt{(}\,\overline{typ_j\ id_j}^{\,j}\,\texttt{)}\,\texttt{\{}\,block\ \texttt{return}\ exp\texttt{;}\,\texttt{\}}\,fdecls \Rightarrow \Theta'} \quad \text{GENM\_TYP}$$

$$\frac{id \notin \Phi \,\texttt{and}\, \Theta \quad \Sigma; cid_{ext}; \Phi; \Theta, id : \texttt{(}\; \overline{typ_j}^{\,j} \;\texttt{)->unit} \vdash fdecls \Rightarrow \Theta'}{\Sigma; cid_{ext}; \Phi; \Theta \vdash \texttt{unit}\; id\; \texttt{(}\overline{typ_j\; id_j}^{\,j}\texttt{)}\; \{ block\; \texttt{return ;} \}\; fdecls \Rightarrow \Theta'} \quad \text{GENM\_UNIT}$$

$\boxed{\Sigma \vdash \mathit{fields}\ \texttt{ok}}$ Σ shows that *fields* is well-formed.

$$\frac{}{\Sigma \vdash \epsilon\ \texttt{ok}} \quad \text{WFF\_NIL}$$

$$\frac{\Sigma \vdash typ \quad \Sigma \vdash \mathit{fields}\ \texttt{ok}}{\Sigma \vdash typ\; id\, \texttt{;}\, \mathit{fields}\ \texttt{ok}} \quad \text{WFF\_CONS}$$

$\boxed{\Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma\texttt{;}\, cid \vdash cinits\ \texttt{ok}}$ Σ, Δ and Γ show that *cinits* is well-formed.

$$\frac{}{\Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma\texttt{;}\, cid \vdash \epsilon\ \texttt{ok}} \quad \text{CINITS\_NIL}$$

$$\frac{\begin{array}{c} cid\; cid_{ext}\{\Phi\texttt{;}\; \overline{typ_j}^{\,j}\;\texttt{;}\Theta\} \in \Sigma \quad id \,\texttt{:}\, typ \in \Phi \\ \Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma\texttt{;}\, -\,\texttt{;}\, typ \vdash_i init\ \texttt{ok} \quad \Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma\texttt{;}\, cid \vdash cinits\ \texttt{ok} \end{array}}{\Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma\texttt{;}\, cid \vdash \texttt{this}\,\texttt{.}\, id\texttt{=}init\texttt{;}\, cinits\ \texttt{ok}} \quad \text{CINITS\_CONS}$$

$\boxed{\Sigma; \Delta; \Gamma; cid \vdash ctor\ \texttt{ok}}$ *ctor* is well-formed.

$$\frac{\begin{array}{l} \Sigma\texttt{;}\,\Delta\texttt{;}\,\cdot\,\texttt{;}\,- \vdash args \Rightarrow \Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma\texttt{;}\,- \\ \Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma\texttt{;}\, cid \vdash cinits\ \texttt{ok} \quad \Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma\texttt{;}\, cid \vdash block \Rightarrow \Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma'\texttt{;}\, cid \end{array}}{\Sigma; \Delta; \Gamma; cid \vdash \texttt{new (}args\texttt{) ()}\; cinits\{block\}\ \texttt{ok}} \quad \text{CTOR\_BASE}$$

$$\frac{\begin{array}{l} \Sigma\texttt{;}\,\Delta\texttt{;}\,\cdot\,\texttt{;}\,- \vdash args \Rightarrow \Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma\texttt{;}\,- \quad cid_1 \texttt{<:} cid_2\{\Phi\texttt{;}\; \overline{typ_j}^{\,j}\;\texttt{;}\Theta\} \in \Sigma \\ cid_2\; cid_{ext\,2}\{\Phi_2\texttt{;}\; \overline{typ'_k}^{\,k}\;\texttt{;}\Theta_2\} \in \Sigma \quad \overline{\Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma'\texttt{;}\,- \vdash exp_k \texttt{ <: } typ'_k}^{\,k} \\ \Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma\texttt{;}\, cid_1 \vdash cinits\ \texttt{ok} \quad \Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma\texttt{;}\, cid_1 \vdash block \Rightarrow \Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma'\texttt{;}\, cid_1 \end{array}}{\Sigma; \Delta; \Gamma; cid_1 \vdash \texttt{new (}args\texttt{) (}\; \overline{exp_k}^{\,k}\;\texttt{)}\; cinits\{block\}\ \texttt{ok}} \quad \text{CTOR\_INHERITANCE}$$

$\boxed{Ctxt \vdash cdecl\ \texttt{ok}}$ *cdecl* is well-formed.

$$\frac{\Sigma \vdash \mathit{fields}\ \texttt{ok} \quad \Sigma; \Delta; \cdot; cid \vdash ctor\ \texttt{ok} \quad \Sigma; \Delta; \cdot; cid \vdash fdecls\ \texttt{ok}}{\Sigma\texttt{;}\,\Delta\texttt{;}\,\Gamma\texttt{;}\,- \vdash \texttt{class}\; cid\; cid_{ext}\{\mathit{fields}\; ctor\; fdecls\}\texttt{;}\ \texttt{ok}} \quad \text{CDECL\_INTRO}$$

$\boxed{\Sigma;\Delta;\cdot;-\vdash \textit{prog} \Rightarrow \mathtt{S'};\mathtt{D'};\cdot;-}$ Collect the global function and class declarations.

$$\frac{}{\Sigma;\Delta;\cdot;-\vdash \epsilon \Rightarrow \mathtt{S'};\mathtt{D'};\cdot;-}\quad \text{FCTXT\_NIL}$$

$$\frac{\Sigma;\Delta;\cdot;-\vdash \textit{prog} \Rightarrow \mathtt{S'};\mathtt{D'};\cdot;-}{\Sigma;\Delta;\cdot;-\vdash \textit{vdecl}\ \textit{prog} \Rightarrow \mathtt{S'};\mathtt{D'};\cdot;-}\quad \text{FCTXT\_VDECL}$$

$$\frac{\begin{array}{l}\textit{cid} \notin \Sigma \quad \textit{cid}_{ext} \in \Sigma \quad \textit{cid}_{ext};\cdot\vdash \textit{fields} \Rightarrow \Phi \quad \Sigma;\textit{cid}_{ext};\Phi;\cdot\vdash \textit{fdecls} \Rightarrow \Theta\\ \Sigma,\ \textit{cid}\ \textit{cid}_{ext}\{\Phi;\ \overline{\textit{typ}_j}^{\,j}\ ;\Theta\};\Delta;\cdot;-\vdash \textit{prog} \Rightarrow \mathtt{S'};\mathtt{D'};\cdot;-\end{array}}{\Sigma;\Delta;\cdot;-\vdash \mathtt{class}\ \textit{cid}\ \textit{cid}_{ext}\{\textit{fields}\ \mathtt{new}\ (\overline{\textit{typ}_j\ \textit{id}_j}^{\,j})\ (\overline{\textit{exp}_m}^{\,m})\ \textit{cinits}\{\textit{block}\}\ \textit{fdecls}\};\ \textit{prog} \Rightarrow \mathtt{S'};\mathtt{D'};\cdot;-}\quad \text{FCTXT\_CDECL}$$

$$\frac{\textit{id} \notin \Delta \quad \Sigma;\Delta,\textit{id}:(\overline{\textit{typ}_j}^{\,j})\mathtt{->}\textit{rtyp};\cdot;-\vdash \textit{prog} \Rightarrow \mathtt{S'};\mathtt{D'};\cdot;-}{\Sigma;\Delta;\cdot;-\vdash \textit{rtyp}\ \textit{id}\ (\overline{\textit{typ}_j\ \textit{id}_j}^{\,j})\ \mathtt{extern}\ \textit{prog} \Rightarrow \mathtt{S'};\mathtt{D'};\cdot;-}\quad \text{FCTXT\_EFUNC}$$

$$\frac{\textit{id} \notin \Delta \quad \Sigma;\Delta,\textit{id}:(\overline{\textit{typ}_j}^{\,j})\mathtt{->}\textit{typ};\cdot;-\vdash \textit{prog} \Rightarrow \mathtt{S'};\mathtt{D'};\cdot;-}{\Sigma;\Delta;\cdot;-\vdash \textit{typ}\ \textit{id}\ (\overline{\textit{typ}_j\ \textit{id}_j}^{\,j})\ \{\textit{block}\ \mathtt{return}\ \textit{exp};\}\ \textit{prog} \Rightarrow \mathtt{S'};\mathtt{D'};\cdot;-}\quad \text{FCTXT\_FUNC\_TYP}$$

$$\frac{\textit{id} \notin \Delta \quad \Sigma;\Delta,\textit{id}:(\overline{\textit{typ}_j}^{\,j})\mathtt{->unit};\cdot;-\vdash \textit{prog} \Rightarrow \mathtt{S'};\mathtt{D'};\cdot;-}{\Sigma;\Delta;\cdot;-\vdash \mathtt{unit}\ \textit{id}\ (\overline{\textit{typ}_j\ \textit{id}_j}^{\,j})\ \{\textit{block}\ \mathtt{return};\}\ \textit{prog} \Rightarrow \mathtt{S'};\mathtt{D'};\cdot;-}\quad \text{FCTXT\_FUNC\_UNIT}$$

$\boxed{\textit{Ctxt} \vdash \textit{prog}\ \mathtt{ok}}$ $\mathtt{S}$, $\mathtt{D}$ and $\mathtt{G}$ show that *prog* is well-formed.

$$\frac{}{\Sigma;\Delta;\cdot;-\vdash \epsilon\ \mathtt{ok}}\quad \text{PROG\_NIL}$$

$$\frac{\Sigma\vdash \textit{typ} \quad \Sigma;\cdot;\cdot;-;\textit{typ} \vdash_i \textit{init}\ \mathtt{ok} \quad \textit{id} \notin \Delta \quad \Sigma;\Delta,\textit{id}:\textit{typ};\cdot;-\vdash \textit{prog}\ \mathtt{ok}}{\Sigma;\Delta;\cdot;-\vdash \textit{typ}\ \textit{id}\mathtt{=}\textit{init};\ \textit{prog}\ \mathtt{ok}}\quad \text{PROG\_VDECL}$$

$$\frac{\Sigma;\Delta;\cdot;-\vdash \textit{fdecl}\ \mathtt{ok} \quad \Sigma;\Delta;\cdot;-\vdash \textit{prog}\ \mathtt{ok}}{\Sigma;\Delta;\cdot;-\vdash \textit{fdecl}\ \textit{prog}\ \mathtt{ok}}\quad \text{PROG\_FDECL}$$

$$\frac{\Sigma;\Delta;\cdot;-\vdash \textit{cdecl}\ \mathtt{ok} \quad \Sigma;\Delta;\cdot;-\vdash \textit{prog}\ \mathtt{ok}}{\Sigma;\Delta;\cdot;-\vdash \textit{cdecl}\ \textit{prog}\ \mathtt{ok}}\quad \text{PROG\_CDECL}$$

$\boxed{\vdash \textit{prog}\ \mathtt{ok}}$ The toplevel program *prog* is closed and well-typed.

$$\frac{\Sigma_{top};\Delta_{top};\cdot;-\vdash \textit{prog} \Rightarrow \mathtt{S'};\mathtt{D'};\cdot;- \quad \Sigma;\Delta;\cdot;-\vdash \textit{prog}\ \mathtt{ok} \quad \text{program}:\mathtt{(int,string[])->int} \in \Delta}{\vdash \textit{prog}\ \mathtt{ok}}\quad \text{TOPLEVEL\_PR}$$