Z, *len*	integer
f	float
n	int
ident, *id*	identifier
label, *l*	label
dcls	global variable declarations
fndefns	function declarations
opt_tid	optional thread id
ef_sig	external signature
p	pointer
typ	
fundef, *fd*	
fn_body	
fn	
ge	
δ	

signedness	::=		Signedness
	|	`Signed`	
	|	`Unsigned`	
intsize	::=		Integer sizes
	|	`I8`	
	|	`I16`	
	|	`I32`	
floatsize	::=		Float sizes
	|	`F32`	
	|	`F64`	
type, *ty*	::=		Types
	|	`void`	the void type
	|	`int (`*intsize*`,`*signedness*`)`	integer types
	|	`float (`*floatsize*`)`	floating-point types
	|	`pointer (`*ty*`)`	pointer types (`*ty`)
	|	`array (`*ty*`,`*len*`)`	array types (`ty[len]`)
	|	`function (`ty^*`,`*ty*`)`	function types
	|	`struct (`*id*`,`ϕ`)`	struct types
	|	`union (`*id*`,`ϕ`)`	union types
	|	`comp_pointer(`*id*`)`	pointer to named struct or union
	|	`(`*ty*`)` S	
typelist, ty^*	::=		Type list
	|	`nil`	
	|	*ty*`::`ty^*	
fieldlist, ϕ	::=		Field lists
	|	`nil`	
	|	`(`*id*`,`*ty*`)::`ϕ	
unary_operation, op_1	::=		unary
	|	`!`	Boolean negation
	|	`~`	Integer complement
	|	`-`	opposite
binary_operation, op_2	::=		binary
	|	`+`	addition
	|	`-`	subtraction
	|	`*`	multiplication
	|	`/`	division
	|	`%`	modulo
	|	`&`	bitwise and
	|	`|`	bitwise or
	|	`^`	bitwise xor
	|	`<<`	left shift
	|	`>>`	right shift

	`==`	equality
	`!=`	not equal
	`<`	less than
	`>`	greater than
	`<=`	less than equal
	`>=`	greater than equal

expr, *e*	::=		typed expression
	|	a^{ty}	expression

expr_descr, *a*	::=		basic expressions
	|	n	integer literal
	|	f	float literal
	|	id	variable
	|	$*e$	unary pointer dereference
	|	$\&e$	address-of
	|	$op_1\ e$	unary operation
	|	$e_1\ op_2\ e_2$	binary operation
	|	$(ty)\,e$	type cast
	|	$e_1 ? e_2 : e_3$	conditional
	|	$e_1 \&\& e_2$	sequential and
	|	$e_1 \mid\mid e_2$	sequential or
	|	`sizeof (`*ty*`)`	size of a type
	|	*e*`.`*id*	access to a member of a struct or union

opt_lhs	::=		optional lhs expression
	|		
	|	(*id*`:`*ty*)`=`	

opt_e	::=		optional expression
	|		
	|	*e*	

e^*	::=		expression list

atomic_statement, *astmt*	::=		atomic
	|	`CAS`	compare and swap
	|	`ATOMIC_INC`	locked inc

statement, *s*	::=		statements
	|	`skip`	do nothing
	|	$e_1 = e_2$	assignment [lvalue = rvalue]
	|	$opt\_lhs\ e'(e^*)$	function or procedure call
	|	$s_1 ; s_2$	sequence
	|	`if (`e_1`) then` s_1 `else` s_2	conditional
	|	`while (`e`) do` s	while
	|	`do` s `while (`e`)`	do while
	|	`for (`$s_1 ; e_2 ; s_3$`)` s	for loop
	|	`break`	break

	`continue`	continue
	`return` *opt_e*	return
	`switch (`*e*`)` *ls*	switch
	l `:` *s*	labelled statement
	`goto` *l*	goto
	`thread_create(`e_1`,`e_2`)`	thread creation
	opt_lhs astmt `(`e^*`)`	atomic operation
	`mfence`	mfence

labeled_statements, *ls*	::=		labeled statements
		`default :` *s*	default
		`case` *n* `:` *s* `;` *ls*	labeled case

arglist	::=		Argument lists
		ty id	
		ty id `,` *arglist*	

varlist	::=		Local variable lists
		ty id `;` *varlist*	

fndefn_internal	::=		function definition
		ty id `(` *arglist* `){` *varlist s* `}`	

program	::=		programs
		dcls fndefns `main =` *id*	

val, *v*	::=		untyped values
		n	integer value
		f	floating point value
		p	pointer
		`undef`	undef

extval, *evl*	::=		external values
		`extint` *n*	external integer value
		`extfloat` *f*	external floating point value

vs	::=		value list
		`nil`	
		v `::` *vs*	
		vs `@[`*v*`]`	

arg_list, *args*	::=		argument lists
		`nil`	
		id^{ty} `::` *args*	

evs	::=		eventval list
		`nil`	

	| $evl :: evs$	
$memory\_chunk,\ c$	::=	
	| `Mint32`	
$mobject\_kind$	::=	
	| `MObjStack`	
$rmw\_instr,\ rmwi$	::=	
	| `ADD` v	
	| `CAS` $v\ v'$	
	| `SET` v	
$mem\_event,\ me$	::=	
	| `write` $p\ memory\_chunk\ v$	
	| `read` $p\ memory\_chunk\ v$	
	| `alloc` $p\ n\ mobject\_kind$	
	| `free` $p\ mobject\_kind$	
	| `rmw` $p\ memory\_chunk\ v\ rmwi$	
	| `fence`	
$event,\ ev$	::=	
	| `call` $id\ evs$	
	| `return` $typ\ evl$	
	| `exit` n	
	| `fail`	
$thread\_event,\ te$	::=	thread events
	| `ext` $event$	externally observable event
	| `mem` mem_event	memory event
	|	thread-local event
	| `exit`	normal exit
	| `start` $p\ vs$	thread start (bootstrap)
opt_pty	::=	optional pointer/type pair
opt_v	::=	optional value
ps	::=	pointer list
$\rho,\ \rho',\ \rho''$	::=	environment
$cont,\ \kappa_s$	::=	statement continuation
	| `stop`	
	| $[\_\ ;\ s_2] \cdot \kappa_s$	sequence
	| $[$`while (`e`) do`$\ s] \cdot \kappa_s$	while
	| $[$`do`$\ s$`while (`e`)`$] \cdot \kappa_s$	do while
	| $[$`for(`$\ ;e_2;\diamond\ s_3)\ s] \cdot \kappa_s$	for loop, pending increment
	| $[$`for(`$\ ;\diamond\ e_2;s_3)\ s] \cdot \kappa_s$	for loop, pending condition evaluation

	$[opt\_lhs\ fd(\_)\,|_\rho] \cdot \kappa_s$	call awaiting args
	$[\texttt{switch}\,\kappa_s]$	switch protecting break
	$[\texttt{free}\ ps\texttt{;}\ \texttt{return}\ opt\_v] \cdot \kappa_s$	

expr_cont, κ_e	::=		expression continuations
		$[op_1^{ty}\,\_] \cdot \kappa_e$	unary operation
		$[\_\ op_2^{ty_1 * ty_2 \to ty}\ e_2] \cdot \kappa_e$	binary operation
		$[v\ op_2^{ty_1 * ty_2 \to ty}\ \_] \cdot \kappa_e$	binary operation
		$[(ty)\_^{ty'}] \cdot \kappa_e$	
		$[\_^{ty}\texttt{?}\,e_2\texttt{:}\,e_3] \cdot \kappa_e$	
		$[\_ \cdot \delta] \cdot \kappa_e$	access to member of struct
		$[\texttt{*}\_^{ty}] \cdot \kappa_e$	load
		$[\_^{ty}\texttt{=}e_2] \cdot \kappa_s$	assignment
		$[v^{ty}\texttt{=}\_] \cdot \kappa_s$	assignment
		$[opt\_lhs\ \_^{ty}\ (e^*)] \cdot \kappa_s$	call function
		$[opt\_lhs\ v^{ty}(vs,\ e^*)] \cdot \kappa_s$	call args
		$[opt\_lhs\ astmt(vs,\ e^*)] \cdot \kappa_s$	atomic args
		$[\texttt{if}\ (\_^{ty})\ \texttt{then}\ s_1\ \texttt{else}\ s_2] \cdot \kappa_s$	if
		$[\texttt{while}\ (\_e)\ \texttt{do}\ s] \cdot \kappa_s$	while
		$[\texttt{do}\ s\ \texttt{while}\ (\_e)] \cdot \kappa_s$	dowhile
		$[\texttt{for}\ (;\ \_e_2;\ s_3)\ s] \cdot \kappa_s$	for loop, pending test
		$[\texttt{return}\ \_] \cdot \kappa_s$	funtion return
		$[\texttt{switch}\ (\_)\ ls] \cdot \kappa_s$	switch
		$[\texttt{thread\_create(\_,}\ e_2\texttt{)}] \cdot \kappa_s$	thread creation
		$[\texttt{thread\_create(}p\texttt{,}\ \_\texttt{)}] \cdot \kappa_s$	thread creation

state, σ	::=		states
		$\texttt{lval}\,(e) \cdot \kappa_e\,|_\rho$	
		$e \cdot \kappa_e\,|_\rho$	
		$v \cdot \kappa_e\,|_\rho$	
		$s \cdot \kappa_s\,|_\rho$	
		$vs \cdot \kappa_s$	
		$\texttt{bind}\,(fn\texttt{,}vs\texttt{,}args) \cdot \kappa_s\,|_\rho$	
		$\texttt{alloc}\,(vs\texttt{,}args) \cdot \kappa_s\,|_\rho$	
		$opt\_lhs\ \texttt{ext(}\_^{\texttt{typ}}\texttt{)} \cdot \kappa_s\,|_\rho$	
		$opt\_lhs\ v \cdot \kappa_s\,|_\rho$	

$\boxed{\sigma \xrightarrow{te} \sigma'}$ Labelled Transitions (parameterised over ge)

$$\frac{}{n^{ty} \cdot \kappa_e\,|_\rho \longrightarrow n \cdot \kappa_e\,|_\rho}\ \textsc{StepConstInt}$$

$$\frac{}{f^{ty} \cdot \kappa_e\,|_\rho \longrightarrow f \cdot \kappa_e\,|_\rho}\ \textsc{StepConstFloat}$$

$$\frac{}{id^{ty} \cdot \kappa_e\,|_\rho \longrightarrow \texttt{lval}\,(id^{ty}) \cdot [\texttt{*}\_^{ty}] \cdot \kappa_e\,|_\rho}\ \textsc{StepVarExprByValue}$$

$$\frac{\rho\texttt{!}id\ \texttt{=}\ \texttt{Some}\ p}{\texttt{lval}\,(id^{ty}) \cdot \kappa_e\,|_\rho \longrightarrow p \cdot \kappa_e\,|_\rho}\ \textsc{StepVarLocal}$$

$$\frac{\rho\texttt{!}id\ \texttt{=}\ \texttt{None} \quad \texttt{Genv.find\_symbol}\ ge\ id\ \texttt{=}\ \texttt{Some}\ p}{\texttt{lval}\,(id^{ty}) \cdot \kappa_e\,|_\rho \longrightarrow p \cdot \kappa_e\,|_\rho}\ \textsc{StepVarGlobal}$$

$$\frac{\texttt{access\_mode}\ ty' = \texttt{By\_value}\ c \quad typ = \texttt{type\_of\_chunk}\ c \quad \texttt{Val.has\_type}\ v\ typ}{p \cdot [*\_^{ty'}] \cdot \kappa_e \,|_\rho \xrightarrow{\texttt{mem (read}\ p\ c\ v\texttt{)}} v \cdot \kappa_e \,|_\rho} \text{ StepLoadByValue}$$

$$\frac{\texttt{access\_mode}\ ty' = \texttt{By\_reference} \setminus\!/ \texttt{access\_mode}\ ty' = \texttt{By\_nothing}}{p \cdot [*\_^{ty'}] \cdot \kappa_e \,|_\rho \longrightarrow p \cdot \kappa_e \,|_\rho} \text{ StepLoadNotByValue}$$

$$\frac{}{\&e^{ty} \cdot \kappa_e \,|_\rho \longrightarrow \texttt{lval}\,(e) \cdot \kappa_e \,|_\rho} \text{ StepAddr}$$

$$\frac{}{e_1 \texttt{?} e_2 : e_3^{ty} \cdot \kappa_e \,|_\rho \longrightarrow e_1 \cdot [\_^{\texttt{typeof}\ e_1 \texttt{?} e_2 : e_3}] \cdot \kappa_e \,|_\rho} \text{ StepEcondition}$$

$$\frac{\texttt{is\_true} v\ ty}{v \cdot [\_^{ty} \texttt{?} e_2 : e_3] \cdot \kappa_e \,|_\rho \longrightarrow e_2 \cdot \kappa_e \,|_\rho} \text{ StepEconditiontrue}$$

$$\frac{\texttt{is\_false} v\ ty}{v \cdot [\_^{ty} \texttt{?} e_2 : e_3] \cdot \kappa_e \,|_\rho \longrightarrow e_3 \cdot \kappa_e \,|_\rho} \text{ StepEconditionfalse}$$

$$\frac{}{*e^{ty} \cdot \kappa_e \,|_\rho \longrightarrow e \cdot [*\_^{ty}] \cdot \kappa_e \,|_\rho} \text{ StepDeref}$$

$$\frac{}{\texttt{lval}\,(*e^{ty}) \cdot \kappa_e \,|_\rho \longrightarrow e \cdot \kappa_e \,|_\rho} \text{ StepDerefLval}$$

$$\frac{}{e.id^{ty} \cdot \kappa_e \,|_\rho \longrightarrow \texttt{lval}\,(e.id^{ty}) \cdot [*\_^{ty}] \cdot \kappa_e \,|_\rho} \text{ StepField}$$

$$\frac{\texttt{typeof}\ e\texttt{=struct}\,(id', \phi) \quad \texttt{field\_offset}\ id\ \phi = \texttt{OK}\ \delta}{\texttt{lval}\,(e.id^{ty}) \cdot \kappa_e \,|_\rho \longrightarrow \texttt{lval}\,(e) \cdot [\_ \cdot \delta] \cdot \kappa_e \,|_\rho} \text{ StepFstruct1}$$

$$\frac{p' = \texttt{MPtr.add}\ p\ (\texttt{Int.repr}\ \delta)}{p \cdot [\_ \cdot \delta] \cdot \kappa_e \,|_\rho \longrightarrow p' \cdot \kappa_e \,|_\rho} \text{ StepFstruct2}$$

$$\frac{\texttt{typeof}\ e\texttt{=union}\,(id', \phi)}{\texttt{lval}\,(e.id^{ty}) \cdot \kappa_e \,|_\rho \longrightarrow \texttt{lval}\,(e) \cdot \kappa_e \,|_\rho} \text{ StepFunion}$$

$$\frac{v\texttt{=Vint}\ (\texttt{Int.repr}\ (\texttt{sizeof}\ ty'))}{\texttt{sizeof}\,(ty')^{ty} \cdot \kappa_e \,|_\rho \longrightarrow v \cdot \kappa_e \,|_\rho} \text{ StepSizeOf}$$

$$\frac{}{op_1\ e^{ty} \cdot \kappa_e \,|_\rho \longrightarrow e \cdot [op_1^{\texttt{typeof}\ e}\ \_] \cdot \kappa_e \,|_\rho} \text{ StepUnop1}$$

$$\frac{\texttt{sem\_unary\_operation}\ op_1\ v\ ty = \texttt{Some}\ v'}{v \cdot [op_1^{ty}\ \_] \cdot \kappa_e \,|_\rho \longrightarrow v' \cdot \kappa_e \,|_\rho} \text{ StepUnop}$$

$$\frac{}{(e_1\ op_2\ e_2)^{ty} \cdot \kappa_e \,|_\rho \longrightarrow e_1 \cdot [\_\ op_2^{\texttt{typeof}\ e_1 * \texttt{typeof}\ e_2 \to ty}\ e_2] \cdot \kappa_e \,|_\rho} \text{ StepBinop1}$$

$$\frac{\texttt{valid\_arg}\ op_2\ ty_1\ ty_2\ v = \texttt{true}}{v \cdot [\_\ op_2^{ty_1 * ty_2 \to ty}\ e_2] \cdot \kappa_e \,|_\rho \longrightarrow e_2 \cdot [v\ op_2^{ty_1 * ty_2 \to ty}\ \_] \cdot \kappa_e \,|_\rho} \text{ StepBinop2}$$

$$\frac{\texttt{sem\_binary\_operation}\ op_2\ v_1\ ty_1\ v_2\ ty_2 = \texttt{Some}\ v}{v_2 \cdot [v_1\ op_2^{ty_1 * ty_2 \to ty}\ \_] \cdot \kappa_e \,|_\rho \longrightarrow v \cdot \kappa_e \,|_\rho} \text{ StepBinop}$$

$$\frac{}{(ty)\,e^{ty'} \cdot \kappa_e \,|_\rho \longrightarrow e \cdot [(ty)\_^{\texttt{typeof}\ e}] \cdot \kappa_e \,|_\rho} \text{ StepCast1}$$

$$\frac{\texttt{cast}\ v\ ty'\ ty\ v'}{v \cdot [(ty)\_^{ty'}] \cdot \kappa_e \,|_\rho \longrightarrow v' \cdot \kappa_e \,|_\rho} \text{ StepCast2}$$

$$\frac{\begin{array}{c} n_0 \texttt{ = Int.repr 0} \\ n_1 \texttt{ = Int.repr 1} \end{array}}{e_1 \texttt{\&\&} e_2{}^{ty} \cdot \kappa_e \,|_\rho \longrightarrow e_1\texttt{?(}e_2\texttt{?(}n_1{}^{ty}\texttt{):(}n_0{}^{ty}\texttt{)}^{ty}\texttt{):}n_0{}^{ty^{ty}} \cdot \kappa_e \,|_\rho} \textsc{StepAndbool}$$

$$\frac{\begin{array}{c} n_0 \texttt{ = Int.repr 0} \\ n_1 \texttt{ = Int.repr 1} \end{array}}{e_1 \texttt{||} e_2{}^{ty} \cdot \kappa_e \,|_\rho \longrightarrow e_1\texttt{?(}n_1{}^{ty}\texttt{):(}e_2\texttt{?(}n_1{}^{ty}\texttt{):(}n_0{}^{ty}\texttt{)}^{ty}\texttt{)}^{ty} \cdot \kappa_e \,|_\rho} \textsc{StepOrbool}$$

$$\frac{}{\texttt{thread\_create(}e_1\texttt{,}e_2\texttt{)} \cdot \kappa_s \,|_\rho \longrightarrow e_1 \cdot [\texttt{thread\_create(\_,}\ e_2\texttt{)}] \cdot \kappa_s \,|_\rho} \textsc{StepThread}$$

$$\frac{}{p \cdot [\texttt{thread\_create(\_,}\ e_2\texttt{)}] \cdot \kappa_s \,|_\rho \longrightarrow e_2 \cdot [\texttt{thread\_create(}p\texttt{,\_)}] \cdot \kappa_s \,|_\rho} \textsc{StepThreadFn}$$

$$\frac{}{v \cdot [\texttt{thread\_create(}p\texttt{,\_)}] \cdot \kappa_s \,|_\rho \xrightarrow{\texttt{start}\ p\ v\texttt{::nil}} \texttt{skip} \cdot \kappa_s \,|_\rho} \textsc{StepThreadEvt}$$

$$\frac{}{e_1\texttt{=}e_2 \cdot \kappa_s \,|_\rho \longrightarrow \texttt{lval (}e_1\texttt{)} \cdot [\texttt{\_}^{\texttt{typeof}\ e_1}\texttt{=}e_2] \cdot \kappa_s \,|_\rho} \textsc{StepAssign1}$$

$$\frac{}{v_1 \cdot [\texttt{\_}^{ty}\texttt{=}e_2] \cdot \kappa_s \,|_\rho \longrightarrow e_2 \cdot [v_1{}^{ty}\texttt{=\_}] \cdot \kappa_s \,|_\rho} \textsc{StepAssign2}$$

$$\frac{\begin{array}{c} \texttt{type\_to\_chunk}\ ty_1 \texttt{ = Some}\ c \\ \texttt{cast\_value\_to\_chunk}\ c\ v_1 \texttt{ =}\ v_2 \end{array}}{v_1 \cdot [p_1{}^{ty_1}\texttt{=\_}] \cdot \kappa_s \,|_\rho \xrightarrow{\texttt{mem (write}\ p_1\ c\ v_2\texttt{)}} \texttt{skip} \cdot \kappa_s \,|_\rho} \textsc{StepAssign}$$

$$\frac{}{s_1\texttt{;}s_2 \cdot \kappa_s \,|_\rho \longrightarrow s_1 \cdot [\texttt{\_;}\ s_2] \cdot \kappa_s \,|_\rho} \textsc{StepSeq}$$

$$\frac{}{opt\_lhs\ e'\texttt{(}e^*\texttt{)} \cdot \kappa_s \,|_\rho \longrightarrow e' \cdot [opt\_lhs\ \texttt{\_}^{\texttt{typeof}\ e'}\ \texttt{(}e^*\texttt{)}] \cdot \kappa_s \,|_\rho} \textsc{StepCall}$$

$$\frac{\begin{array}{c} \texttt{Genv.find\_funct ge}\ v \texttt{ = Some}\ fd \\ \texttt{type\_of\_fundef}\ fd \texttt{ =}\ ty \end{array}}{v \cdot [opt\_lhs\ \texttt{\_}^{ty}\ \texttt{(nil)}] \cdot \kappa_s \,|_\rho \longrightarrow \texttt{nil} \cdot [opt\_lhs\ fd\texttt{(\_)}\,|_\rho] \cdot \kappa_s} \textsc{StepCallargsnone}$$

$$\frac{}{v \cdot [opt\_lhs\ \texttt{\_}^{ty}\ \texttt{(}e\texttt{::}e^*\texttt{)}] \cdot \kappa_s \,|_\rho \longrightarrow e \cdot [opt\_lhs\ v^{ty}\texttt{(nil,}\ e^*\texttt{)}] \cdot \kappa_s \,|_\rho} \textsc{StepCallArgs1}$$

$$\frac{}{v_1 \cdot [opt\_lhs\ v^{ty}\texttt{(}vs\texttt{,}\ e\texttt{::}e^*\texttt{)}] \cdot \kappa_s \,|_\rho \longrightarrow e \cdot [opt\_lhs\ v^{ty}\texttt{(}vs\texttt{@[}v_1\texttt{],}\ e^*\texttt{)}] \cdot \kappa_s \,|_\rho} \textsc{StepCallArgs2}$$

$$\frac{\begin{array}{c} \texttt{Genv.find\_funct ge}\ v \texttt{ = Some}\ fd \\ \texttt{type\_of\_fundef}\ fd \texttt{ =}\ ty \end{array}}{v' \cdot [opt\_lhs\ v^{ty}\texttt{(}vs\texttt{,}\ \texttt{nil)}] \cdot \kappa_s \,|_\rho \longrightarrow vs\texttt{@[}v'\texttt{]} \cdot [opt\_lhs\ fd\texttt{(\_)}\,|_\rho] \cdot \kappa_s} \textsc{StepCallFinish}$$

$$\frac{}{opt\_lhs\ astmt\texttt{(}e\texttt{::}e^*\texttt{)} \cdot \kappa_s \,|_\rho \longrightarrow e \cdot [opt\_lhs\ astmt\texttt{(nil,}\ e^*\texttt{)}] \cdot \kappa_s \,|_\rho} \textsc{StepAtomic}$$

$$\frac{}{v \cdot [opt\_lhs\ astmt\texttt{(}vs\texttt{,}\ e\texttt{::}e^*\texttt{)}] \cdot \kappa_s \,|_\rho \longrightarrow e \cdot [opt\_lhs\ astmt\texttt{(}vs\texttt{@[}v\texttt{],}\ e^*\texttt{)}] \cdot \kappa_s \,|_\rho} \textsc{StepAtomicArgs}$$

$$\frac{\begin{array}{l} \texttt{sem\_atomic\_statement}\ astmt\ \texttt{(}\ vs\ \texttt{++}\ v\ \texttt{::}\ \texttt{nil}\ \texttt{)} \texttt{ = Some (}p\texttt{,}\ rmwi\texttt{)} \\ \texttt{Val.has\_type v' (type\_of\_chunk Mint32)} \end{array}}{v \cdot [\ astmt\texttt{(}vs\texttt{,}\ \texttt{nil)}] \cdot \kappa_s \,|_\rho \xrightarrow{\texttt{mem (rmw}\ p\ \texttt{Mint32}\ v'\ rmwi\texttt{)}} \texttt{skip} \cdot \kappa_s \,|_\rho} \textsc{StepAtomicFinishNone}$$

$$\frac{\begin{array}{l} \texttt{sem\_atomic\_statement}\ astmt\ \texttt{(}\ vs\ \texttt{++}\ v\ \texttt{::}\ \texttt{nil}\ \texttt{)} \texttt{ = Some (}p\texttt{,}\ rmwi\texttt{)} \\ \texttt{Val.has\_type v' (type\_of\_chunk Mint32)} \end{array}}{v \cdot [(id\texttt{:}ty\texttt{)=}\ astmt\texttt{(}vs\texttt{,}\ \texttt{nil)}] \cdot \kappa_s \,|_\rho \xrightarrow{\texttt{mem (rmw}\ p\ \texttt{Mint32}\ v'\ rmwi\texttt{)}} (id\texttt{:}ty\texttt{)=}\ v' \cdot \kappa_s \,|_\rho} \textsc{StepAtomicFinishSome}$$

$$\frac{}{\texttt{mfence} \cdot \kappa_s \,|_\rho \xrightarrow{\texttt{mem fence}} \texttt{skip} \cdot \kappa_s \,|_\rho} \textsc{StepFence}$$

$$\frac{}{\texttt{continue} \cdot [\_\,;\, s] \cdot \kappa_s \,|_\rho \;\longrightarrow\; \texttt{continue} \cdot \kappa_s \,|_\rho} \quad \textsc{StepContinue}$$

$$\frac{}{\texttt{break} \cdot [\_\,;\, s] \cdot \kappa_s \,|_\rho \;\longrightarrow\; \texttt{break} \cdot \kappa_s \,|_\rho} \quad \textsc{StepBreak}$$

$$\frac{}{\texttt{if}\,(e)\,\texttt{then}\,s_1\,\texttt{else}\,s_2 \cdot \kappa_s \,|_\rho \;\longrightarrow\; e \cdot [\texttt{if}\,(\_^{\texttt{typeof}\,e})\,\texttt{then}\,s_1\,\texttt{else}\,s_2] \cdot \kappa_s \,|_\rho} \quad \textsc{StepIfThenElse}$$

$$\frac{\texttt{is\_true}\, v\; ty}{v \cdot [\texttt{if}\,(\_^{ty})\,\texttt{then}\,s_1\,\texttt{else}\,s_2] \cdot \kappa_s \,|_\rho \;\longrightarrow\; s_1 \cdot \kappa_s \,|_\rho} \quad \textsc{StepIfThenElseTrue}$$

$$\frac{\texttt{is\_false}\, v\; ty}{v \cdot [\texttt{if}\,(\_^{ty})\,\texttt{then}\,s_1\,\texttt{else}\,s_2] \cdot \kappa_s \,|_\rho \;\longrightarrow\; s_2 \cdot \kappa_s \,|_\rho} \quad \textsc{StepIfThenElseFalse}$$

$$\frac{}{\texttt{while}\,(e)\,\texttt{do}\,s \cdot \kappa_s \,|_\rho \;\longrightarrow\; e \cdot [\texttt{while}\,(\_e)\,\texttt{do}\,s] \cdot \kappa_s \,|_\rho} \quad \textsc{StepWhile}$$

$$\frac{\texttt{is\_true}\, v\;(\texttt{typeof}\, e)}{v \cdot [\texttt{while}\,(\_e)\,\texttt{do}\,s] \cdot \kappa_s \,|_\rho \;\longrightarrow\; s \cdot [\texttt{while}\,(e)\,\texttt{do}\,s] \cdot \kappa_s \,|_\rho} \quad \textsc{StepWhileTrue}$$

$$\frac{\texttt{is\_false}\, v\;(\texttt{typeof}\, e)}{v \cdot [\texttt{while}\,(\_e)\,\texttt{do}\,s] \cdot \kappa_s \,|_\rho \;\longrightarrow\; \texttt{skip} \cdot \kappa_s \,|_\rho} \quad \textsc{StepWhileFalse}$$

$$\frac{}{\texttt{continue} \cdot [\texttt{while}\,(e)\,\texttt{do}\,s] \cdot \kappa_s \,|_\rho \;\longrightarrow\; \texttt{while}\,(e)\,\texttt{do}\,s \cdot \kappa_s \,|_\rho} \quad \textsc{StepContinueWhile}$$

$$\frac{}{\texttt{break} \cdot [\texttt{while}\,(e)\,\texttt{do}\,s] \cdot \kappa_s \,|_\rho \;\longrightarrow\; \texttt{skip} \cdot \kappa_s \,|_\rho} \quad \textsc{StepBreakWhile}$$

$$\frac{}{\texttt{do}\,s\,\texttt{while}\,(e) \cdot \kappa_s \,|_\rho \;\longrightarrow\; s \cdot [\texttt{do}\,s\,\texttt{while}\,(e)] \cdot \kappa_s \,|_\rho} \quad \textsc{StepDoWhile}$$

$$\frac{\texttt{is\_true}\, v\;(\texttt{typeof}\, e)}{v \cdot [\texttt{do}\;s\;\texttt{while}\,(\_e)] \cdot \kappa_s \,|_\rho \;\longrightarrow\; \texttt{do}\,s\,\texttt{while}\,(e) \cdot \kappa_s \,|_\rho} \quad \textsc{StepDoWhileTrue}$$

$$\frac{\texttt{is\_false}\, v\;(\texttt{typeof}\, e)}{v \cdot [\texttt{do}\;s\;\texttt{while}\,(\_e)] \cdot \kappa_s \,|_\rho \;\longrightarrow\; \texttt{skip} \cdot \kappa_s \,|_\rho} \quad \textsc{StepDoWhileFalse}$$

$$\frac{}{\texttt{continue} \cdot [\texttt{do}\,s\,\texttt{while}\,(e)] \cdot \kappa_s \,|_\rho \;\longrightarrow\; e \cdot [\texttt{do}\;s\;\texttt{while}\,(\_e)] \cdot \kappa_s \,|_\rho} \quad \textsc{StepDoContinueWhile}$$

$$\frac{}{\texttt{break} \cdot [\texttt{do}\,s\,\texttt{while}\,(e)] \cdot \kappa_s \,|_\rho \;\longrightarrow\; \texttt{skip} \cdot \kappa_s \,|_\rho} \quad \textsc{StepDoBreakWhile}$$

$$\frac{}{\texttt{for}\,(s_1\,;e_2\,;s_3)\,s \cdot \kappa_s \,|_\rho \;\longrightarrow\; s_1 \cdot [\texttt{for}(\;;\diamond\, e_2\,;s_3)\;\; s] \cdot \kappa_s \,|_\rho} \quad \textsc{StepForInit}$$

$$\frac{}{\texttt{skip} \cdot [\texttt{for}(\;;\diamond\, e_2\,;s_3)\;\; s] \cdot \kappa_s \,|_\rho \;\longrightarrow\; e_2 \cdot [\texttt{for}\;(;\_{e_2};\; s_3)\;\; s] \cdot \kappa_s \,|_\rho} \quad \textsc{StepForCond}$$

$$\frac{\texttt{is\_true}\, v\;(\texttt{typeof}\, e_2)}{v \cdot [\texttt{for}\;(;\_{e_2};\; s_3)\;\; s] \cdot \kappa_s \,|_\rho \;\longrightarrow\; s \cdot [\texttt{for}(\;;e_2\,;\diamond\, s_3)\;\; s] \cdot \kappa_s \,|_\rho} \quad \textsc{StepForTrue}$$

$$\frac{\texttt{is\_false}\, v\;(\texttt{typeof}\, e_2)}{v \cdot [\texttt{for}\;(;\_{e_2};\; s_3)\;\; s] \cdot \kappa_s \,|_\rho \;\longrightarrow\; \texttt{skip} \cdot \kappa_s \,|_\rho} \quad \textsc{StepForFalse}$$

$$\frac{}{\texttt{skip} \cdot [\texttt{for}(\;;e_2\,;\diamond\, s_3)\;\; s] \cdot \kappa_s \,|_\rho \;\longrightarrow\; s_3 \cdot [\texttt{for}(\;;\diamond\, e_2\,;s_3)\;\; s] \cdot \kappa_s \,|_\rho} \quad \textsc{StepForIncr}$$

$$\frac{}{\texttt{break} \cdot [\texttt{for}(\;;e_2\,;\diamond\, s_3)\;\; s] \cdot \kappa_s \,|_\rho \;\longrightarrow\; \texttt{skip} \cdot \kappa_s \,|_\rho} \quad \textsc{StepForBreak}$$

$$\frac{}{\texttt{continue} \cdot [\texttt{for}(\;;e_2\,;\diamond\, s_3)\;\; s] \cdot \kappa_s \,|_\rho \;\longrightarrow\; s_3 \cdot [\texttt{for}(\;;\diamond\, e_2\,;s_3)\;\; s] \cdot \kappa_s \,|_\rho} \quad \textsc{StepForContinue}$$

$$\frac{\begin{array}{l}\texttt{call\_cont}\ \kappa_s = (\texttt{Kcall None (Internal}\ fn)\ \rho''\ \kappa_s')\\ fn.(\texttt{fn\_return}) = \texttt{Tvoid}\\ ps = \texttt{sorted\_pointers\_of\_env}\ \rho'\end{array}}{\texttt{return} \cdot \kappa_s\,|_{\rho'} \longrightarrow \texttt{skip} \cdot [\texttt{free}\ ps; \texttt{return None}] \cdot \kappa_s\,|_{\rho'}}\ \textsc{StepReturnNone}$$

$$\frac{}{\texttt{skip} \cdot [\texttt{free}\ p :: ps; \texttt{return}\ opt\_v] \cdot \kappa_s\,|_\rho \xrightarrow{\texttt{mem (free}\ p\ \texttt{MObjStack)}} \texttt{skip} \cdot [\texttt{free}\ ps; \texttt{return}\ opt\_v] \cdot \kappa_s\,|_\rho}\ \textsc{StepRet}$$

$$\frac{\begin{array}{l}\texttt{call\_cont}\ \kappa_s = \kappa_s'\\ \texttt{get\_fundef}\ \kappa_s' = \texttt{Some (Internal}\ fn)\\ fn.(\texttt{fn\_return}) <> \texttt{Tvoid}\end{array}}{\texttt{return}\ e \cdot \kappa_s\,|_{\rho'} \longrightarrow e \cdot [\texttt{return}\ \_] \cdot \kappa_s\,|_{\rho'}}\ \textsc{StepReturnSome}$$

$$\frac{ps = \texttt{sorted\_pointers\_of\_env}\ \rho}{v \cdot [\texttt{return}\ \_] \cdot \kappa_s\,|_\rho \longrightarrow \texttt{skip} \cdot [\texttt{free}\ ps; \texttt{return (Some}\ v)] \cdot \kappa_s\,|_\rho}\ \textsc{StepReturnSome1}$$

$$\frac{}{\texttt{switch (}e\texttt{)}\ ls \cdot \kappa_s\,|_\rho \longrightarrow e \cdot [\texttt{switch (\_)}\ ls] \cdot \kappa_s\,|_\rho}\ \textsc{StepSwitch}$$

$$\frac{s = \texttt{seq\_of\_labeled\_statement (select\_switch}\ n\ ls)}{n \cdot [\texttt{switch (\_)}\ ls] \cdot \kappa_s\,|_\rho \longrightarrow s \cdot [\texttt{switch}\ \kappa_s]\,|_\rho}\ \textsc{StepSelectSwitch}$$

$$\frac{}{\texttt{break} \cdot [\texttt{switch}\ \kappa_s]\,|_\rho \longrightarrow \texttt{skip} \cdot \kappa_s\,|_\rho}\ \textsc{StepBreakSwitch}$$

$$\frac{}{\texttt{continue} \cdot [\texttt{switch}\ \kappa_s]\,|_\rho \longrightarrow \texttt{continue} \cdot \kappa_s\,|_\rho}\ \textsc{StepContinueSwitch}$$

$$\frac{}{l : s \cdot \kappa_s\,|_\rho \longrightarrow s \cdot \kappa_s\,|_\rho}\ \textsc{StepLabel}$$

$$\frac{\begin{array}{l}\texttt{call\_cont}\ \kappa_s = \kappa_s'\\ \texttt{get\_fundef}\ \kappa_s' = \texttt{(Some (Internal}\ fn\texttt{))}\\ \texttt{find\_label}\ l\ fn.(\texttt{fn\_body})\ \kappa_s' = \texttt{Some (}s', \kappa_s''\texttt{)}\end{array}}{\texttt{goto}\ l \cdot \kappa_s\,|_\rho \longrightarrow s' \cdot \kappa_s''\,|_\rho}\ \textsc{StepGoto}$$

$$\frac{\begin{array}{l}args = fn.(\texttt{fn\_params}) \texttt{ ++ } fn.(\texttt{fn\_vars})\\ fd = \texttt{Internal}\ fn\end{array}}{vs \cdot [opt\_lhs\ fd(\_)\,|_\rho] \cdot \kappa_s \longrightarrow \texttt{alloc (}vs\texttt{,}args\texttt{)} \cdot [opt\_lhs\ fd(\_)\,|_\rho] \cdot \kappa_s\,|_{\rho_{\text{empty}}}}\ \textsc{StepFunctionInternal}$$

$$\frac{n = \texttt{Int.repr(sizeof}\ ty\texttt{)}}{\texttt{alloc (}vs\texttt{,}id^{ty}\texttt{::}args\texttt{)} \cdot \kappa_s\,|_\rho \xrightarrow{\texttt{mem (alloc}\ p\ n\ \texttt{MObjStack)}} \texttt{alloc (}vs\texttt{,}args\texttt{)} \cdot \kappa_s\,|_{\rho \oplus (id \mapsto p)}}\ \textsc{StepAllocLocal}$$

$$\frac{\begin{array}{l}args = fn.(\texttt{fn\_params})\\ fd = \texttt{(Internal}\ fn\texttt{)}\end{array}}{\texttt{alloc (}vs\texttt{,nil)} \cdot [opt\_lhs\ fd(\_)\,|_{\rho'}] \cdot \kappa_s\,|_{\rho''} \longrightarrow \texttt{bind (}fn\texttt{,}vs\texttt{,}args\texttt{)} \cdot [opt\_lhs\ fd(\_)\,|_{\rho'}] \cdot \kappa_s\,|_{\rho''}}\ \textsc{StepBindArgsS}$$

$$\frac{\begin{array}{l}\rho!id = \texttt{Some}\ p\\ \texttt{type\_to\_chunk}\ ty = \texttt{(Some}\ c\texttt{)}\\ \texttt{cast\_value\_to\_chunk}\ c\ v_1 = v_2\end{array}}{\texttt{bind (}fn\texttt{,}v_1\texttt{::}vs\texttt{,}id^{ty}\texttt{::}args\texttt{)} \cdot \kappa_s\,|_\rho \xrightarrow{\texttt{mem (write}\ p\ c\ v_2\texttt{)}} \texttt{bind (}fn\texttt{,}vs\texttt{,}args\texttt{)} \cdot \kappa_s\,|_\rho}\ \textsc{StepBindArgs}$$

$$\frac{s{=}fn.(\texttt{fn\_body})}{\texttt{bind (}fn\texttt{,nil,nil)} \cdot \kappa_s\,|_\rho \longrightarrow s \cdot \kappa_s\,|_\rho}\ \textsc{StepTransferFun}$$

$$\frac{\begin{array}{l}\texttt{true (* event\_match (external\_function id targs ty) vs t vres -> *)}\\ fd = \texttt{External}\ id\ ty^*\ ty\\ vs = \texttt{map val\_of\_eval}\ evs\end{array}}{vs \cdot [opt\_lhs\ fd(\_)\,|_\rho] \cdot \kappa_s \xrightarrow{\texttt{ext (call}\ id\ evs\texttt{)}} opt\_lhs\ \texttt{ext(\_}^{\texttt{typ}}\texttt{)} \cdot \kappa_s\,|_\rho}\ \textsc{StepExternalCall}$$

$$\frac{\begin{array}{l}\texttt{Val.has\_type}\ v\ typ\\ fd\ \texttt{=}\ \texttt{External}\ id\ ty^*\ ty\\ typ\ \texttt{=}\ \texttt{match}\ (\texttt{opttyp\_of\_type}\ ty)\ \texttt{with}\ \mid\ \texttt{Some}\ \texttt{x}\ \texttt{=>}\ \texttt{x}\ \mid\ \texttt{None}\ \texttt{=>}\ \texttt{Ast.Tint}\ \texttt{end}\\ v\ \texttt{=}\ \texttt{val\_of\_eval}\ evl\end{array}}{opt\_lhs\ \texttt{ext(\_}^{\texttt{typ}}\texttt{)} \cdot \kappa_\mathrm{s}\,|_\rho \xrightarrow{\texttt{ext (return}\ typ\ evl\texttt{)}} opt\_lhs\ v \cdot \kappa_\mathrm{s}\,|_\rho}\ \textsc{StepExternalRet}$$

$$\frac{\begin{array}{l}\rho\texttt{!}id\ \texttt{=}\ \texttt{Some}\ p\\ \texttt{type\_to\_chunk}\ ty\ \texttt{=}\ \texttt{Some}\ c\\ \texttt{cast\_value\_to\_chunk}\ c\ v_1\ \texttt{=}\ v_2\end{array}}{(id\texttt{:}ty)\texttt{=}\ v_1 \cdot \kappa_\mathrm{s}\,|_\rho \xrightarrow{\texttt{mem (write}\ p\ c\ v_2\texttt{)}} \texttt{skip} \cdot \kappa_\mathrm{s}\,|_\rho}\ \textsc{StepExternalStoreSomeLocal}$$

$$\frac{\begin{array}{l}\rho\texttt{!}id\ \texttt{=}\ \texttt{None}\\ \texttt{Genv.find\_symbol ge}\ id\ \texttt{=}\ \texttt{Some}\ p\\ \texttt{type\_to\_chunk}\ ty\ \texttt{=}\ \texttt{Some}\ c\\ \texttt{cast\_value\_to\_chunk}\ c\ v_1\ \texttt{=}\ v_2\end{array}}{(id\texttt{:}ty)\texttt{=}\ v_1 \cdot \kappa_\mathrm{s}\,|_\rho \xrightarrow{\texttt{mem (write}\ p\ c\ v_2\texttt{)}} \texttt{skip} \cdot \kappa_\mathrm{s}\,|_\rho}\ \textsc{StepExternalStoreSomeGlobal}$$

$$\frac{}{v \cdot \kappa_\mathrm{s}\,|_\rho \longrightarrow \texttt{skip} \cdot \kappa_\mathrm{s}\,|_\rho}\ \textsc{StepExternalStoreNone}$$

$$\frac{}{\texttt{skip} \cdot [\_\texttt{;}\ s_2] \cdot \kappa_\mathrm{s}\,|_\rho \longrightarrow s_2 \cdot \kappa_\mathrm{s}\,|_\rho}\ \textsc{StepSkip}$$

$$\frac{}{\texttt{skip} \cdot [\texttt{while}\ (e)\ \texttt{do}\ s] \cdot \kappa_\mathrm{s}\,|_\rho \longrightarrow \texttt{while}\ (e)\ \texttt{do}\ s \cdot \kappa_\mathrm{s}\,|_\rho}\ \textsc{StepWhileLoop}$$

$$\frac{}{\texttt{skip} \cdot [\texttt{do}\ s\ \texttt{while}\ (e)] \cdot \kappa_\mathrm{s}\,|_\rho \longrightarrow e \cdot [\texttt{do}\ s\ \texttt{while}\ (\_e)] \cdot \kappa_\mathrm{s}\,|_\rho}\ \textsc{StepDoWhileLoop}$$

$$\frac{}{\texttt{skip} \cdot [\texttt{switch}\ \kappa_\mathrm{s}]\,|_\rho \longrightarrow \texttt{skip} \cdot \kappa_\mathrm{s}\,|_\rho}\ \textsc{StepSkipSwitch}$$

$$\frac{\texttt{call\_cont}\ \kappa_\mathrm{s}\ \texttt{=}\ [\ fd(\_)\,|_{\rho'}] \cdot \kappa'_\mathrm{s}}{\texttt{skip} \cdot [\texttt{free nil; return}\ opt\_v] \cdot \kappa_\mathrm{s}\,|_{\rho''} \longrightarrow \texttt{skip} \cdot \kappa'_\mathrm{s}\,|_{\rho'}}\ \textsc{StepReturnNoneFinish}$$

$$\frac{\begin{array}{l}\texttt{type\_to\_chunk}\ ty\ \texttt{=}\ (\texttt{Some}\ c)\\ \rho'\texttt{!}id\ \texttt{=}\ \texttt{Some}\ p\\ \texttt{call\_cont}\ \kappa_\mathrm{s}\ \texttt{=}\ [(id\texttt{:}ty)\texttt{=}\ fd(\_)\,|_{\rho'}] \cdot \kappa'_\mathrm{s}\\ \texttt{cast\_value\_to\_chunk}\ c\ v_1\ \texttt{=}\ v_2\end{array}}{\texttt{skip} \cdot [\texttt{free nil; return (Some}\ v_1\texttt{)}] \cdot \kappa_\mathrm{s}\,|_{\rho''} \xrightarrow{\texttt{mem (write}\ p\ c\ v_2\texttt{)}} \texttt{skip} \cdot \kappa'_\mathrm{s}\,|_{\rho'}}\ \textsc{StepReturnSomeFinishLocal}$$

$$\frac{\begin{array}{l}\texttt{type\_to\_chunk}\ ty\ \texttt{=}\ (\texttt{Some}\ c)\\ \rho'\texttt{!}id\ \texttt{=}\ \texttt{None}\\ \texttt{Genv.find\_symbol ge}\ id\ \texttt{=}\ \texttt{Some}\ p\\ \texttt{call\_cont}\ \kappa_\mathrm{s}\ \texttt{=}\ [(id\texttt{:}ty)\texttt{=}\ fd(\_)\,|_{\rho'}] \cdot \kappa'_\mathrm{s}\\ \texttt{cast\_value\_to\_chunk}\ c\ v_1\ \texttt{=}\ v_2\end{array}}{\texttt{skip} \cdot [\texttt{free nil; return (Some}\ v_1\texttt{)}] \cdot \kappa_\mathrm{s}\,|_{\rho''} \xrightarrow{\texttt{mem (write}\ p\ c\ v_2\texttt{)}} \texttt{skip} \cdot \kappa'_\mathrm{s}\,|_{\rho'}}\ \textsc{StepReturnSomeFinishGlobal}$$

$$\frac{}{\texttt{skip} \cdot \texttt{stop}\,|_\rho \xrightarrow{\texttt{exit}} \texttt{skip} \cdot \texttt{stop}\,|_\rho}\ \textsc{StepStop}$$

```
Definition rules:          94 good    0 bad
Definition rule clauses:  178 good    0 bad
```