x, y, *ident*	OCaml type variable for symbols
tyvar_TY	OCaml type variable for types
ty_act	annotated C type
tag	struct/union tag
k	OCaml fixed-width integer
natval	OCaml arbitrary-width natural number
	OCaml C source location type
n, i	
<*impl-const*>	
intval	memory integer value
memval	
member	C struct/union member name
τ	C type
annots	
Mem_mem_iv_constraint	
ub-name	
string	
n	
bool	
memory-order	
linux-memory-order	
thread-id	

bTy	::=		Core base types
		`unit`	unit
		bool	boolean
		`integer`	integer
		`real`	rational numbers?
		`loc`	location
		$[bTy]$	list
		$(bTy_1, .., bTy_n)$	tuple
		`struct` *tag*	
		$\{bTy\}$	set
		`opt` (bTy)	option
		$bTy_1, .., bTy_n \to bTy$	parameter types
binop	::=		binary operators
		`+`	
		`-`	
		`*`	
		`/`	
		`rem_t`	
		`rem_f`	
		`^`	
		`=`	
		`>`	
		`<`	
		`>=`	
		`<=`	
		`/\`	
		`\/`	
polarity	::=		memory action polarities
		`Pos`	sequenced by `let weak` and `let strong`
		`Neg`	only sequenced by `let strong`
ident	::=		Core identifier
		ident	
name	::=		
		ident	Core identifier
		<*impl-const*>	implementation-defined constant
ptrval	::=		pointers
		`nullptr`	
		`funcptr` *ident*	
		`concptr` *natval*	
object_value	::=		C object values
		intval	integer value

	$\vert$	$ptrval$	pointer value
	$\vert$	$\texttt{array}\,(loaded\_value_1,\,..\,,loaded\_value_n)$	C array value
	$\vert$	$(\,\texttt{struct}\,tag)\{\,\overline{.member_i : \tau_i = memval_i}^{\,i}\,\}$	C struct value
	$\vert$	$(\,\texttt{union}\,tag)\{.member = memval\}$	C union value
$loaded\_value$	$::=$		potentially unspecified C object value
	$\vert$	$\texttt{specified}\,(object\_value)$	non-unspecified loaded value
τ	$::=$		base type
	$\vert$	bTy	
$value$	$::=$		Core values
	$\vert$	$object\_value$	C object value
	$\vert$	$loaded\_value$	loaded C object value
	$\vert$	$\texttt{Unit}$	
	$\vert$	$\texttt{True}$	
	$\vert$	$\texttt{False}$	
	$\vert$	$[value_1, .., value_i]$	
	$\vert$	$(value_1,\,..\,,value_i)$	tuple
$ctor$	$::=$		data constructors
	$\vert$	$\texttt{Nil}\,\tau$	empty list
	$\vert$	$\texttt{Cons}$	list cons
	$\vert$	$\texttt{Tuple}$	tuple
	$\vert$	$\texttt{Array}$	C array
	$\vert$	$\texttt{Ivmax}$	max integer value
	$\vert$	$\texttt{Ivmin}$	min integer value
	$\vert$	$\texttt{Ivsizeof}$	sizeof value
	$\vert$	$\texttt{Ivalignof}$	alignof value
	$\vert$	$\texttt{IvCOMPL}$	bitwise complement
	$\vert$	$\texttt{IvAND}$	bitwise AND
	$\vert$	$\texttt{IvOR}$	bitwise OR
	$\vert$	$\texttt{IvXOR}$	bitwise XOR
	$\vert$	$\texttt{Specified}$	non-unspecified loaded value
	$\vert$	$\texttt{Unspecified}$	unspecified loaded value
	$\vert$	$\texttt{Fvfromint}$	cast integer to floating value
	$\vert$	$\texttt{Ivfromfloat}$	cast floating to integer value
$maybesym\_base\_type$	$::=$		
	$\vert$	$\_ : bTy$	
	$\vert$	$ident : bTy$	
$mu\_pattern\_aux$	$::=$		
	$\vert$	$maybesym\_base\_type$	
	$\vert$	$ctor(\,\overline{mu\_pattern_i}^{\,i}\,)$	
$mu\_pattern$	$::=$		

	|	*annots mu_pattern_aux*	
mu_sym_or_pattern	::=		
	|	*ident*	
	|	*mu_pattern*	
code_asym	::=		annotated symbol
	|	*ident*	
mu_pexpr_aux	::=		Core pure express
	|	*ident*	
	|	`impl_const`	implementation
	|	*value*	
	|	`constrained` $(\overline{Mem\_mem\_iv\_constraint_i, code\_asym_i}^{\,i})$	constrained val
	|	`undef` $(ub\text{-}name)$	undefined beha
	|	`error` $(string, code\_asym)$	impl-defined sta
	|	$ctor(\overline{code\_asym_i}^{\,i})$	data constructo
	|	`array_shift` $(code\_asym_1, \tau, code\_asym_2)$	pointer array sh
	|	`member_shift` $(code\_asym, ident, member)$	pointer struct/u
	|	`not` $(code\_asym)$	boolean not
	|	$code\_asym_1\ binop\ code\_asym_2$	
	|	$($`struct` $ident)\{\overline{.member_i = code\_asym_i}^{\,i}\}$	C struct expres
	|	$($`union` $ident)\{.member = code\_asym\}$	C union express
	|	`memberof` $(ident, member, code\_asym)$	C struct/union
	|	$name(code\_asym_1, .., code\_asym_n)$	pure function c
	|	`assert_undef` $(code\_asym, , ub\text{-}name)$	
	|	`bool_to_integer` $(code\_asym)$	
	|	`conv_int` $(\tau, code\_asym)$	
	|	`wrapI` $(\tau, code\_asym)$	
e	::=		
	|	`code_annots` $tyvar\_TY\ mu\_pexpr\_aux_1, .., mu\_pexpr\_aux_n$	
mu_tpexpr_aux	::=		Core top-level pur
	|	`case` $code\_asym$ `of` $\overline{\mid mu\_pattern_i => \texttt{mu\_tpexpri}}^{\,i}$ `end`	pattern matchin
	|	`let` $mu\_sym\_or\_pattern = mu\_tpexpr1 \in mu\_tpexpr2$	pure let
	|	`if` $code\_asym$ `then` $mu\_tpexpr1$ `else` $mu\_tpexpr2$	pure if
	|	`done` $code\_asym$	pure done
mu_action_aux	::=		memory actions
	|	`create` (e_1, e_2)	
	|	`create_readonly` (e_1, e_2, e_3)	
	|	`alloc` (e_1, e_2)	
	|	`kill` $(bool, e)$	the boolean ind
	|	`store` $(bool, e_1, e_2, e_3, memory\text{-}order)$	the boolean ind
	|	`load` $(e_1, e_2, memory\text{-}order)$	
	|	`rmw` $(e_1, e_2, e_3, e_4, memory\text{-}order_1, memory\text{-}order_2)$	

	|	$\texttt{fence}\,(\mathit{memory\text{-}order})$
	|	$\texttt{compare\_exchange\_strong}\,(e_1, e_2, e_3, e_4, \mathit{memory\text{-}order}_1, \mathit{memory\text{-}order}_2)$
	|	$\texttt{compare\_exchange\_weak}\,(e_1, e_2, e_3, e_4, \mathit{memory\text{-}order}_1, \mathit{memory\text{-}order}_2)$
	|	$\texttt{linux\_fence}\,(\mathit{linux\text{-}memory\text{-}order})$
	|	$\texttt{linux\_load}\,(e_1, e_2, \mathit{linux\text{-}memory\text{-}order})$
	|	$\texttt{linux\_store}\,(e_1, e_2, e_3, \mathit{linux\text{-}memory\text{-}order})$
	|	$\texttt{linux\_rmw}\,(e_1, e_2, e_3, \mathit{linux\text{-}memory\text{-}order})$
$\mathit{mu\_action}$	::=	
	|	$\mathit{mu\_action\_aux}$
$\mathit{mu\_paction}$	::=	
	|	$\mathit{polarity}\ \mathit{mu\_action}$
	|	$\mathit{mu\_action}$
	|	$\neg\,(\mathit{mu\_action})$
memop	::=	
	|	$\mathit{pointer\text{-}equality\text{-}operator}$
	|	$\mathit{pointer\text{-}relational\text{-}operator}$
	|	`ptrdiff`
	|	`intFromPtr`
	|	`ptrFromInt`
	|	`ptrValidForDeref`
	|	`ptrWellAligned`
	|	`ptrArrayShift`
	|	`memcpy`
	|	`memcmp`
	|	`realloc`
	|	`va_start`
	|	`va_copy`
	|	`va_arg`
	|	`va_end`
$\mathit{code\_sym\_base\_type\_pair}$	::=	
	|	$\texttt{code\_sym}\ :\ \mathit{bTy}$
$\mathit{base\_type\_pexpr\_pair}$	::=	
	|	$\mathit{bTy} := e$
E	::=	
	|	$\texttt{pure}\,(e)$
	|	$\texttt{memop}\,(\mathit{memop}, e_1, \,..\,, e_n)$
	|	$\mathit{mu\_paction}$
	|	$\texttt{case}\ e\ \texttt{with}\ \overline{\mid \mathit{mu\_pattern}_i => E_i}^{\,i}\ \texttt{end}$
	|	$\texttt{let}\ \mathit{mu\_pattern} = e\ \in\ E$
	|	$\texttt{if}\ e\ \texttt{then}\ E_1\ \texttt{else}\ E_2$
	|	`skip`

	$\vert$	`ccall` $(e_1, e_2, \overline{e_i}^{\,i})$		C functi
	$\vert$	`pcall` $(name, \overline{e_i}^{\,i})$		Core pro
	$\vert$	`unseq` $(E_1, .., E_n)$		unsequen
	$\vert$	`let weak` $mu\_pattern = E_1 \in E_2$		weak sec
	$\vert$	`let strong` $mu\_pattern = E_1 \in E_2$		strong se
	$\vert$	`let atomic` $code\_sym\_base\_type\_pair = mu\_action_1 \in mu\_paction_2$		atomic s
	$\vert$	`indet` $[n](E)$		indeterm
	$\vert$	`bound` $[n](E)$		...and b
	$\vert$	`nd` $(E_1, .., E_n)$		nondeter
	$\vert$	`save` $code\_sym\_base\_type\_pair(\overline{\texttt{code\_symi} : base\_type\_pexpr\_pair_i}^{\,i}) \in E$		save labe
	$\vert$	`run code_sym` $(\overline{e_i}^{\,i})$		run from
	$\vert$	`par` $(E_1, .., E_n)$		cppmem
	$\vert$	`wait` $(thread\text{-}id)$		wait for
E	::=			
	$\vert$	$annots\, E$		
$terminals$	::=			
	$\vert$	λ		
	$\vert$	$\longrightarrow$		
	$\vert$	$\rightarrow$		
	$\vert$	$\vdash$		
	$\vert$	$\in$		
	$\vert$	Π		
	$\vert$	$\forall$		
	$\vert$	$\multimap$		
	$\vert$	$\supset$		
	$\vert$	Σ		
	$\vert$	$\exists$		
	$\vert$	$\star$		
	$\vert$	$\wedge$		
	$\vert$	$\bigwedge$		
	$\vert$	$\neg$		
	$\vert$	$=$		
bt	::=			OCaml typ
	$\vert$			
$bool$	::=			
	$\vert$	`true`		
	$\vert$	`false`		
z	::=			OCaml arb
	$\vert$	`of_intval` $intval$	M	
	$\vert$	`of_nat` $natval$	M	
lit	::=			

	$\vert$	$ident$		
	$\vert$	$()$		
	$\vert$	$bool$		
	$\vert$	$\texttt{int}\, z$		
	$\vert$	$\texttt{ptr}\, z$		
$bool\_op$	$::=$			
	$\vert$	$\neg\, index\_term$		
	$\vert$	$index\_term_1 = index\_term_2$		
	$\vert$	$\bigwedge(index\_term_1, \ldots, index\_term_n)$		
$list\_op$	$::=$			
	$\vert$	$[index\_term_1, \ldots, index\_term_n]$		
	$\vert$	$index\_term^{(k)}$		
$tuple\_op$	$::=$			
	$\vert$	$(index\_term_1, \ldots, index\_term_n)$		
	$\vert$	$index\_term^{(k)}$		
$pointer\_op$	$::=$			
	$\vert$	$\texttt{nullop}$		
$param\_op$	$::=$			
	$\vert$	$index\_term(index\_term_1, \ldots, index\_term_n)$		
$index\_term\_aux$	$::=$			
	$\vert$	$bool\_op$		
	$\vert$	$list\_op$		
	$\vert$	$pointer\_op$		
	$\vert$	$param\_op$		
$index\_term$	$::=$			
	$\vert$	lit		
	$\vert$	$index\_term\_aux\ bt$		
	$\vert$	$(index\_term)$	S	parentheses
	$\vert$	$index\_term[index\_term_1/ident_1, \ldots, index\_term_n/ident_n]$	M	
arg	$::=$			argument types
	$\vert$	$\Pi\, ident : bTy.arg$		
	$\vert$	$\forall\, ident : \texttt{logSort}.arg$		
	$\vert$	$\texttt{resource} \multimap arg$		
	$\vert$	$index\_term \supset arg$		
	$\vert$	$\texttt{I}$		
ret	$::=$			return types
	$\vert$	$\Sigma\, ident : bTy.ret$		
	$\vert$	$\exists\, ident : \texttt{logSort}.ret$		

$|$ $\texttt{resource} \star \mathit{ret}$
$|$ $\mathit{index\_term} \wedge \mathit{ret}$
$|$ $\texttt{I}$

Γ ::= computational var env
$|$ $\texttt{empty}$
$|$ $\Gamma, x : \mathit{bTy}$

Λ ::= logical var env
$|$ $\texttt{empty}$
$|$ Λ, x

Ξ ::= constraints env
$|$ $\texttt{empty}$
$|$ $\Xi, \texttt{phi}$

$\mathit{formula}$ ::=
$|$ $\mathit{judgement}$
$|$ $\texttt{not}\,(\mathit{formula})$
$|$ $\mathit{ident} : \mathit{bTy} \in \Gamma$
$|$ $\mathit{formula}_1 \quad .. \quad \mathit{formula}_n$

Jtype ::=
$|$ $\Gamma; \Lambda; \Xi \vdash \mathit{value} : \mathit{ident}, \mathit{bTy}, \mathit{index\_term}$
$|$ $\Gamma; \Lambda; \Xi \vdash \mathit{mu\_pexpr\_aux} : \mathit{ret}$

$\mathit{judgement}$ ::=
$|$ Jtype

$\mathit{user\_syntax}$ ::=
$|$ x
$|$ $\mathit{tyvar\_TY}$
$|$ $\mathit{ty\_act}$
$|$ tag
$|$ k
$|$ natval
$|$
$|$ n
$|$ <*impl-const*>
$|$ intval
$|$ memval
$|$ member
$|$ τ
$|$ annots
$|$ $\mathit{Mem\_mem\_iv\_constraint}$
$|$ $\mathit{ub\text{-}name}$
$|$ string

$|$
$|$ n
$|$ $bool$
$|$
$|$ $memory\text{-}order$
$|$ $linux\text{-}memory\text{-}order$
$|$ $thread\text{-}id$
$|$ bTy
$|$ $binop$
$|$ $polarity$
$|$ $ident$
$|$ $name$
$|$ $ptrval$
$|$ $object\_value$
$|$ $loaded\_value$
$|$ τ
$|$ $value$
$|$ $ctor$
$|$ $maybesym\_base\_type$
$|$ $mu\_pattern\_aux$
$|$ $mu\_pattern$
$|$ $mu\_sym\_or\_pattern$
$|$ $code\_asym$
$|$ mu_pexpr_aux
$|$ e
$|$ mu_tpexpr_aux
$|$ mu_action_aux
$|$ mu_action
$|$ $mu\_paction$
$|$ $memop$
$|$ $code\_sym\_base\_type\_pair$
$|$ $base\_type\_pexpr\_pair$
$|$ E
$|$ E
$|$ $terminals$
$|$ bt
$|$ $bool$
$|$ z
$|$ lit
$|$ $bool\_op$
$|$ $list\_op$
$|$ $tuple\_op$
$|$ $pointer\_op$
$|$ $param\_op$
$|$ $index\_term\_aux$
$|$ $index\_term$

$$
\begin{array}{ll}
| & \mathit{arg} \\
| & \mathit{ret} \\
| & \Gamma \\
| & \Lambda \\
| & \Xi \\
| & \mathit{formula}
\end{array}
$$

$\boxed{\Gamma; \Lambda; \Xi \vdash \mathit{value} : \mathit{ident}, \mathit{bTy}, \mathit{index\_term}}$

$$\frac{}{\Gamma; \Lambda; \Xi \vdash \mathit{intval} : y, \mathtt{integer}, y = \mathtt{int\,of\_intval}\ \mathit{intval}} \quad \textsc{Val\_Obj\_Int}$$

$$\frac{}{\Gamma; \Lambda; \Xi \vdash \mathtt{nullptr} : y, \mathtt{loc}, y = \mathtt{nullop}} \quad \textsc{Val\_Obj\_Ptr\_Null}$$

$$\frac{}{\Gamma; \Lambda; \Xi \vdash \mathtt{funcptr}\ \mathit{ident} : y, \mathtt{loc}, y = \mathit{ident}} \quad \textsc{Val\_Obj\_Ptr\_Func}$$

$$\frac{}{\Gamma; \Lambda; \Xi \vdash \mathtt{concptr}\ \mathit{natval} : y, \mathtt{loc}, y = \mathtt{ptr\,of\_nat}\ \mathit{natval}} \quad \textsc{Val\_Obj\_Ptr\_Conc}$$

$$\frac{\Gamma; \Lambda; \Xi \vdash \mathit{loaded\_value}_1 : y_1, \mathit{bTy}, \mathit{index\_term}_1 \quad .. \quad \Gamma; \Lambda; \Xi \vdash \mathit{loaded\_value}_n : y_n, \mathit{bTy}, \mathit{index\_ter}}{\Gamma; \Lambda; \Xi \vdash \mathtt{array}\,(\mathit{loaded\_value}_1, \,..\,, \mathit{loaded\_value}_n) : y, \mathtt{integer} \to \mathit{bTy}, \bigwedge(\mathit{index\_term}_1, \,..\,, \mathit{index\_term}_n)\,[y(\mathtt{int}\ z)}$$

$$\frac{}{\Gamma; \Lambda; \Xi \vdash \mathtt{Unit} : y, \mathtt{unit}, y = ()} \quad \textsc{Val\_Unit}$$

$$\frac{}{\Gamma; \Lambda; \Xi \vdash \mathtt{True} : y, \mathit{bool}, y = \mathtt{true}} \quad \textsc{Val\_True}$$

$$\frac{}{\Gamma; \Lambda; \Xi \vdash \mathtt{False} : y, \mathit{bool}, y = \mathtt{false}} \quad \textsc{Val\_False}$$

$$\frac{\Gamma; \Lambda; \Xi \vdash \mathit{value}_1 : y_1, \mathit{bTy}, \mathit{index\_term}_1 \quad .. \quad \Gamma; \Lambda; \Xi \vdash \mathit{value}_n : y_n, \mathit{bTy}, \mathit{index\_term}_n}{\Gamma; \Lambda; \Xi \vdash [\mathit{value}_1, .., \mathit{value}_i] : y, [\mathit{bTy}], (\bigwedge(\mathit{index\_term}_1, \,..\,, \mathit{index\_term}_n))[y^{(k)}/y_1, \,..\,, y^{(k)}/y_n]} \quad \textsc{Val\_List}$$

$$\frac{\Gamma; \Lambda; \Xi \vdash \mathit{value}_1 : y_1, \mathit{bTy}_1, \mathit{index\_term}_1 \quad .. \quad \Gamma; \Lambda; \Xi \vdash \mathit{value}_n : y_n, \mathit{bTy}_n, \mathit{index\_term}_n}{\Gamma; \Lambda; \Xi \vdash (\mathit{value}_1, \,..\,, \mathit{value}_n) : y, (\mathit{bTy}_1, \,..\,, \mathit{bTy}_n), \bigwedge(\mathit{index\_term}_1, \,..\,, \mathit{index\_term}_n)\,[y^{(k)}/y_1, \,..\,, y^{(k)}/y_n]} \quad \textsc{Val\_T}$$

$\boxed{\Gamma; \Lambda; \Xi \vdash \mathit{mu\_pexpr\_aux} : \mathit{ret}}$

$$\frac{x : \mathit{bTy} \in \Gamma}{\Gamma; \Lambda; \Xi \vdash x : \Sigma\, y : \mathit{bTy}.\mathtt{I}} \quad \textsc{PExpr\_Var}$$

$$\frac{\Gamma; \Lambda; \Xi \vdash \mathit{value} : y, \mathit{bTy}, \mathit{index\_term}}{\Gamma; \Lambda; \Xi \vdash \mathit{value} : \Sigma\, y : \mathit{bTy}.\mathit{index\_term} \wedge \mathtt{I}} \quad \textsc{PExpr\_Val}$$

$$\frac{x : \mathit{bool} \in \Gamma}{\Gamma; \Lambda; \Xi \vdash \mathtt{not}\,(x) : \Sigma\, y : \mathit{bool}.y = (\neg\, x) \wedge \mathtt{I}} \quad \textsc{PExpr\_Not}$$

```
Definition rules:        13 good    0 bad
Definition rule clauses: 19 good    0 bad
```