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
	|	$bTy_1, .., bTy_n \rightarrow bTy$	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

	|	*ptrval*	pointer value
	|	$\texttt{array}\,(loaded\_value_1,\,..,loaded\_value_n)$	C array value
	|	$(\,\texttt{struct}\,tag)\{\,\overline{.member_i : \tau_i = memval_i}^{\,i}\,\}$	C struct value
	|	$(\,\texttt{union}\,tag)\{.member = memval\}$	C union value
loaded_value	::=		potentially unsp
	|	$\texttt{specified}\,(object\_value)$	non-unspecifie
τ	::=		base type
	|	bTy	
value	::=		Core values
	|	*object_value*	C object valu
	|	*loaded_value*	loaded C obje
	|	`Unit`	
	|	`True`	
	|	`False`	
	|	$[value_1, .., value_i]$	
	|	$(value_1,\,..,value_i)$	tuple
ctor	::=		data constructor
	|	$\texttt{Nil}\,\tau$	empty list
	|	`Cons`	list cons
	|	`Tuple`	tuple
	|	`Array`	C array
	|	`Ivmax`	max integer v
	|	`Ivmin`	min integer va
	|	`Ivsizeof`	sizeof value
	|	`Ivalignof`	alignof value
	|	`IvCOMPL`	bitwise compl
	|	`IvAND`	bitwise AND
	|	`IvOR`	bitwise OR
	|	`IvXOR`	bitwise XOR
	|	`Specified`	non-unspecifie
	|	`Unspecified`	unspecified lo
	|	`Fvfromint`	cast integer to
	|	`Ivfromfloat`	cast floating t
maybesym_base_type	::=		
	|	$\_ : bTy$	binders = {}
	|	$ident : bTy$	binders = *ident*
mu_pattern_aux	::=		
	|	*maybesym_base_type*	
	|	$ctor(\,\overline{mu\_pattern_i}^{\,i}\,)$	
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 valu
	| `undef` (*ub-name*)	undefined beha
	| `error` (*string*, *code_asym*)	impl-defined sta
	| *ctor*($\overline{code\_asym_i}^{\,i}$)	data constructo
	| `array_shift` (*code_asym*$_1$, τ, *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 expres
	| `memberof` (*ident*, *member*, *code_asym*)	C struct/union
	| *name*(*code_asym*$_1$, .. , *code_asym*$_n$)	pure function c
	| `assert_undef` (*code_asym*, , *ub-name*)	
	| `bool_to_integer` (*code_asym*)	
	| `conv_int` (τ, *code_asym*)	
	| `wrapI` (τ, *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` $\overline{\vert mu\_pattern_i$ `=>` `mu_tpexpri`$}^{\,i}$ `end`	pattern matchi
	| `let` *mu_sym_or_pattern* = *mu_tpexpr*1 $\in$ *mu_tpexpr*2	pure let
	| `if` *code_asym* `then` *mu_tpexpr*1 `else` *mu_tpexpr*2	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-order*)	the boolean ind
	| `load` (e_1, e_2, *memory-order*)	
	| `rmw` (e_1, e_2, e_3, e_4, *memory-order*$_1$, *memory-order*$_2$)	

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

	$\texttt{ccall}\,(e_1, e_2, \overline{e_i}^{\,i})$	C functi
	$\texttt{pcall}\,(name, \overline{e_i}^{\,i})$	Core pro
	$\texttt{unseq}\,(E_1, .., E_n)$	unsequer
	$\texttt{let}\,\texttt{weak}\,mu\_pattern = E_1 \in E_2$	weak seq
	$\texttt{let}\,\texttt{strong}\,mu\_pattern = E_1 \in E_2$	strong s
	$\texttt{let}\,\texttt{atomic}\,code\_sym\_base\_type\_pair = mu\_action_1 \in mu\_paction_2$	atomic s
	$\texttt{indet}\,[n](E)$	indeterm
	$\texttt{bound}\,[n](E)$	...and b
	$\texttt{nd}\,(E_1, .., E_n)$	nondeter
	$\texttt{save}\,code\_sym\_base\_type\_pair(\overline{\texttt{code\_symi} : base\_type\_pexpr\_pair_i}^{\,i}) \in E$	save lab
	$\texttt{run}\,\texttt{code\_sym}\,(\overline{e_i}^{\,i})$	run from
	$\texttt{par}\,(E_1, .., E_n)$	cppmem
	$\texttt{wait}\,(thread\text{-}id)$	wait for

E	::=	
	$annots\,E$	

terminals	::=	
	λ	
	$\longrightarrow$	
	$\rightarrow$	
	$\vdash$	
	$\in$	
	Π	
	$\forall$	
	$\multimap$	
	$\supset$	
	Σ	
	$\exists$	
	$\star$	
	$\wedge$	
	$\bigwedge$	
	$\neg$	
	$=$	

bt	::=	OCaml typ

bool	::=	
	$\texttt{true}$	
	$\texttt{false}$	

z	::=	OCaml arb
	$\texttt{of\_intval}\,intval$	M
	$\texttt{of\_nat}\,natval$	M

lit	::=	

	$ident$		
	$()$		
	$bool$		
	$\texttt{int}\, z$		
	$\texttt{ptr}\, z$		

$bool\_op$	$::=$		
	$\neg\, index\_term$		
	$index\_term_1 = index\_term_2$		
	$\bigwedge(index\_term_1, .., index\_term_n)$		

$list\_op$	$::=$		
	$[index\_term_1, .., index\_term_n]$		
	$index\_term^{(k)}$		

$tuple\_op$	$::=$		
	$(index\_term_1, .., index\_term_n)$		
	$index\_term^{(k)}$		

$pointer\_op$	$::=$		
	$\texttt{nullop}$		

$param\_op$	$::=$		
	$index\_term(index\_term_1, .., index\_term_n)$		

$index\_term\_aux$	$::=$		
	$bool\_op$		
	$list\_op$		
	$pointer\_op$		
	$param\_op$		

$index\_term$	$::=$		
	lit		
	$index\_term\_aux\ bt$		
	$(index\_term)$	S	parentheses
	$index\_term[index\_term_1/ident_1, .., index\_term_n/ident_n]$	M	

arg	$::=$		argument types
	$\Pi\, ident : bTy.arg$		
	$\forall\, ident : \texttt{logSort}\, .arg$		
	$\texttt{resource} \multimap arg$		
	$index\_term \supset arg$		
	$\texttt{I}$		

ret	$::=$	return types
	$\Sigma\, ident : bTy.ret$	
	$\exists\, ident : \texttt{logSort}\, .ret$	

	|	`resource` $\star$ *ret*
	|	*index_term* $\wedge$ *ret*
	|	`I`
Γ	::=	computational var env
	|	`empty`
	|	$\Gamma, x : bTy$
Λ	::=	logical var env
	|	`empty`
	|	Λ, x
Ξ	::=	constraints env
	|	`empty`
	|	Ξ, `phi`
formula	::=	
	|	*judgement*
	|	`not` (*formula*)
	|	$ident : bTy \in \Gamma$
	|	$formula_1 \quad .. \quad formula_n$
Jtype	::=	
	|	$\Gamma; \Lambda; \Xi \vdash value : ident, bTy, index\_term$
	|	$\Gamma; \Lambda; \Xi \vdash mu\_pexpr\_aux : ret$
judgement	::=	
	|	*Jtype*
user_syntax	::=	
	|	x
	|	$tyvar\_TY$
	|	ty_act
	|	tag
	|	k
	|	$natval$
	|	
	|	n
	|	<*impl-const*>
	|	$intval$
	|	$memval$
	|	$member$
	|	τ
	|	$annots$
	|	$Mem\_mem\_iv\_constraint$
	|	*ub-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
| $bool\_op$
| $list\_op$
| $tuple\_op$
| $pointer\_op$
| $param\_op$
| $index\_term\_aux$
| $index\_term$

| *arg*
| *ret*
| Γ
| Λ
| Ξ
| *formula*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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