$ident$, x, y, y_p, v, v_p, _	v is for values, subscript p is for pointers, rest are identifiers
$impl\_const$	implementation-defined constant
mem_int	memory integer value
$member$	C struct/union member name
$annots$	annotations
nat	OCaml arbitrary-width natural number
n, i	index variables
loc	OCaml type for C source
mem_iv_c	OCaml type for memory constraints on integer values
UB_name	undefined behaviour
$string$	OCaml string
$tyvar\_TY$	OCaml type variable for types
	OCaml type for symbol prefix
mem_order, _	OCaml type for memory order
$linux\_mem\_order$	OCaml type for Linux memory order
k	OCaml fixed-width integer
bt,	Ott-hack, ignore

Sctypes_t, τ	::=			C type
	|	$\tau*$		pointer to type τ
tag	::=			OCaml type for struct/union tag
	|	*ident*		
β	::=			base types
	|	`unit`		unit
	|	`bool`		boolean
	|	`integer`		integer
	|	`real`		rational numbers?
	|	`loc`		location
	|	`array` β		array
	|	$[\beta]$		list
	|	$(\beta_1, .., \beta_n)$		tuple
	|	`struct` *tag*		struct
	|	$\{\beta\}$		set
	|	`opt` (β)		option
	|	$\beta_1, .., \beta_n \to \beta$		parameter types
	|	of_ctype(τ)	M	of a C type
binop	::=			binary operators
	|	`+`		addition
	|	`-`		subtraction
	|	`*`		multiplication
	|	`/`		division
	|	`rem_t`		modulus
	|	`rem_f`		remainder
	|	`^`		exponentiation
	|	`=`		equality, defined both for integer and C types

	$\mid$	`>`	greater than
	$\mid$	`<`	less than
	$\mid$	`>=`	greater than or equal to
	$\mid$	`<=`	less than or equal to
	$\mid$	`/\`	conjucntion
	$\mid$	`\/`	disjunction
polarity	::=		memory action polarities
	$\mid$	`Pos`	sequenced by `let weak` and `let strong`
	$\mid$	`Neg`	only sequenced by `let strong`
name	::=		
	$\mid$	*ident*	Core identifier
	$\mid$	*impl_const*	implementation-defined constant
mem_ptr	::=		pointer values
	$\mid$	`nullptr`	null pointer
	$\mid$	`funcptr` *ident*	function pointer
	$\mid$	`concptr` *nat*	concrete pointer
mem_val	::=		memory value
	$\mid$	`int` *mem_int*	
	$\mid$	*mem_ptr*	
	$\mid$	`array` $(\overline{mem\_val_i}^{\,i})$	
	$\mid$	$($`struct` $ident)\{\overline{.member_i = mem\_val_i}^{\,i}\}$	
	$\mid$	`union` *ident member*	
object_value	::=		C object values (inhabitants of object types), which can be read/stored
	$\mid$	*mem_int*	integer value
	$\mid$	*mem_ptr*	pointer value

	$\mid$	$\texttt{array}\,(\,\overline{\mathit{loaded\_value}_i}^{\,i}\,)$	C array value
	$\mid$	$(\,\texttt{struct}\,\mathit{ident})\{\,\overline{.\mathit{member}_i : \tau_i = \mathit{mem\_val}_i}^{\,i}\,\}$	C struct value
	$\mid$	$(\,\texttt{union}\,\mathit{ident})\{.\mathit{member} = \mathit{mem\_val}\}$	C union value
$\mathit{loaded\_value}$	$::=$		potentially unspecified C object values
	$\mid$	$\texttt{specified}\,\mathit{object\_value}$	specified loaded value
value	$::=$		Core values
	$\mid$	$\mathit{object\_value}$	C object value
	$\mid$	$\mathit{loaded\_value}$	loaded C object value
	$\mid$	`Unit`	unit
	$\mid$	`True`	boolean true
	$\mid$	`False`	boolean false
	$\mid$	$\beta[\mathit{value}_1, \ldots, \mathit{value}_i]$	list
	$\mid$	$(\mathit{value}_1, \ldots, \mathit{value}_i)$	tuple
ctor	$::=$		data constructors
	$\mid$	$\texttt{Nil}\,\beta$	empty list
	$\mid$	`Cons`	list cons
	$\mid$	`Tuple`	tuple
	$\mid$	`Array`	C array
	$\mid$	`Ivmax`	max integer value
	$\mid$	`Ivmin`	min integer value
	$\mid$	`Ivsizeof`	sizeof value
	$\mid$	`Ivalignof`	alignof value
	$\mid$	`IvCOMPL`	bitwise complement
	$\mid$	`IvAND`	bitwise AND
	$\mid$	`IvOR`	bitwise OR
	$\mid$	`IvXOR`	bitwise XOR
	$\mid$	`Specified`	non-unspecified loaded value

	$\mid$	`Fvfromint`		cast integer to floating value
	$\mid$	`Ivfromfloat`		cast floating to integer value
$ident\_opt\_\beta$	$::=$		type annotated optional identifier	
	$\mid$	$\_ : \beta$		
	$\mid$	$ident : \beta$		
$pattern\_aux$	$::=$			
	$\mid$	$ident\_opt\_\beta$		
	$\mid$	$ctor(\overline{pattern_i}^{\,i})$		
$pattern$	$::=$			
	$\mid$	$loc\ annots\ pattern\_aux$		
$ident\_or\_pattern$	$::=$			
	$\mid$	$ident$		
	$\mid$	$pattern$		
$pexpr\_aux$	$::=$		pure expressions	
	$\mid$	$ident$	[V]	
	$\mid$	$impl\_const$	[V]	implementation-defined constant
	$\mid$	$value$	[V]	
	$\mid$	$\texttt{constrained}\,(\overline{mem\_iv\_c_i, ident_i}^{\,i})$	[V]	constrained value
	$\mid$	$\texttt{error}\,(string, ident)$	[V]	impl-defined static error
	$\mid$	$ctor(\overline{ident_i}^{\,i})$		data constructor application
	$\mid$	$\texttt{array\_shift}\,(ident_1, \tau, ident_2)$		pointer array shift
	$\mid$	$\texttt{member\_shift}\,(ident, ident, member)$		pointer struct/union member shift
	$\mid$	$\texttt{not}\,(ident)$		boolean not
	$\mid$	$ident_1\ binop\ ident_2$		binary operations
	$\mid$	$(\,\texttt{struct}\ ident)\{\overline{.member_i = ident_i}^{\,i}\}$	[V]	C struct expression

	$\mid$	$($ `union` $\mathit{ident})\{.\mathit{member} = \mathit{ident}\}$		[V] C union expression
	$\mid$	`memberof` $(\mathit{ident}, \mathit{member}, \mathit{ident})$		C struct/union member access
	$\mid$	$\mathit{name}(\mathit{ident}_1, .., \mathit{ident}_n)$		pure function call
	$\mid$	`assert_undef` $(\mathit{ident}, \mathit{loc}, \mathit{UB\_name})$		
	$\mid$	`bool_to_integer` (ident)		
	$\mid$	`conv_int` (τ, ident)		
	$\mid$	`wrapI` (τ, ident)		
pexpr	::=			pure expressions with location and annotations
	$\mid$	$\mathit{loc}\ \mathit{annots}\ \mathit{tyvar\_TY}\ \mathit{pexpr\_aux}$		[V]
$\mathit{tpexpr\_aux}$	::=			top-level pure expressions
	$\mid$	`undef` $\mathit{loc}\ \mathit{UB\_name}$		[V] undefined behaviour
	$\mid$	`case` ident `of` $\overline{\mid \mathit{pattern}_i \Rightarrow \mathit{tpexpr}_i}^{i}$ `end`		pattern matching
	$\mid$	`let` $\mathit{ident\_or\_pattern} = \mathit{pexpr}$ `in` tpexpr		pure let
	$\mid$	`if` ident `then` tpexpr_1 `else` tpexpr_2		pure if
	$\mid$	`done` ident		[V] pure done
tpexpr	::=			pure top-level pure expressions with location and annotations
	$\mid$	$\mathit{loc}\ \mathit{annots}\ \mathit{tyvar\_TY}\ \mathit{tpexpr\_aux}$		[V]
	$\mid$	$[\mathcal{C}/\mathcal{C}']\mathit{tpexpr}$	M	
$\mathit{m\_kill\_kind}$	::=			
	$\mid$	`dynamic`		
	$\mid$	`static` τ		
bool, _	::=			OCaml booleans
	$\mid$	`true`		
	$\mid$	`false`		

action_aux	::=		memory actions
	|	`create` $(ident, \tau)$	
	|	`create_readonly` $(ident_1, \tau, ident_2)$	
	|	`alloc` $(ident_1, ident_2)$	
	|	`kill` $(m\_kill\_kind, ident)$	
	|	`store` $(bool, \tau, ident_1, ident_2, mem\_order)$	true means store is locking
	|	`load` $(\tau, ident, mem\_order)$	
	|	`rmw` $(\tau, ident_1, ident_2, ident_3, mem\_order_1, mem\_order_2)$	
	|	`fence` (mem_order)	
	|	`cmp_exch_strong` $(\tau, ident_1, ident_2, ident_3, mem\_order_1, mem\_order_2)$	
	|	`cmp_exch_weak` $(\tau, ident_1, ident_2, ident_3, mem\_order_1, mem\_order_2)$	
	|	`linux_fence` $(linux\_mem\_order)$	
	|	`linux_load` $(\tau, ident, linux\_mem\_order)$	
	|	`linux_store` $(\tau, ident_1, ident_2, linux\_mem\_order)$	
	|	`linux_rmw` $(\tau, ident_1, ident_2, linux\_mem\_order)$	
action	::=		
	|	$loc\, action\_aux$	
memop	::=		operations involving the memory state
	|	$ident_1 == ident_2$	pointer equality comparison
	|	$ident_1 \neq ident_2$	pointer inequality comparison
	|	$ident_1 < ident_2$	pointer less-than comparison
	|	$ident_1 > ident_2$	pointer greater-than comparison
	|	$ident_1 \leq ident_2$	pointer less-than comparison
	|	$ident_1 \geq ident_2$	pointer greater-than comparison
	|	$ident_1 -_\tau ident_2$	pointer subtraction
	|	`intFromPtr` $(\tau_1, \tau_2, ident)$	cast of pointer value to integer value
	|	`ptrFromInt` $(\tau_1, \tau_2, ident)$	cast of integer value to pointer value
	|	`ptrValidForDeref` $(\tau, ident)$	dereferencing validity predicate

	$\mid$	`ptrWellAligned` (τ, ident)		
	$\mid$	`ptrArrayShift` $(\mathit{ident}_1, \tau, \mathit{ident}_2)$		
	$\mid$	`memcpy` $(\mathit{ident}_1, \mathit{ident}_2, \mathit{ident}_3)$		
	$\mid$	`memcmp` $(\mathit{ident}_1, \mathit{ident}_2, \mathit{ident}_3)$		
	$\mid$	`realloc` $(\mathit{ident}_1, \mathit{ident}_2, \mathit{ident}_3)$		
	$\mid$	`va_start` $(\mathit{ident}_1, \mathit{ident}_2)$		
	$\mid$	`va_copy` (ident)		
	$\mid$	`va_arg` (ident, τ)		
	$\mid$	`va_end` (ident)		
paction	::=			memory actions with polarity
	$\mid$	*action*	M	positive, sequenced by both `let weak` and `let strong`
	$\mid$	$\neg\,(\mathit{action})$	M	negative, only sequenced by `let strong`
expr_aux	::=			(effectful) expressions
	$\mid$	`pure` (pexpr)	[V]	pure expression
	$\mid$	`memop` (memop)		pointer op involving memory
	$\mid$	*paction*		memory action
	$\mid$	`skip`	[V]	skip
	$\mid$	`ccall` $(\tau, \mathit{ident}, \overline{\mathit{ident}_i}^{\,i})$		C function call
	$\mid$	`pcall` $(\mathit{name}, \overline{\mathit{ident}_i}^{\,i})$		procedure call
expr	::=			(effectful) expressions with location and annotations
	$\mid$	*loc annots expr_aux*	[V]	
texpr_aux	::=			top-level expressions
	$\mid$	`let` $\mathit{ident\_or\_pattern} = \mathit{pexpr}$ `in` texpr		
	$\mid$	`let weak` $\mathit{pattern} = \mathit{expr}$ `in` texpr		weak sequencing
	$\mid$	`let strong` $\mathit{ident\_or\_pattern} = \mathit{expr}$ `in` texpr		strong sequencing
	$\mid$	`case` ident `with` $\overline{\mid \mathit{pattern}_i \Rightarrow \mathit{texpr}_i}^{\,i}$ `end`		pattern matching

	$\vert$	$\texttt{if}\ ident\ \texttt{then}\ texpr_1\ \texttt{else}\ texpr_2$		conditional
	$\vert$	$\texttt{unseq}\,(expr_1,\ ..,expr_n)$	[V]	unsequenced expressions
	$\vert$	$\texttt{nd}\,(texpr_1,\ ..,texpr_n)$		nondeterministic sequencing
	$\vert$	$\texttt{done}\ ident$	[V]	end of top-level expression
	$\vert$	$\texttt{undef}\ loc\ UB\_name$	[V]	undefined behaviour
	$\vert$	$\texttt{run}\ ident\ ident_1,\ ..,ident_n$		run from label
$texpr$	::=			top-level expressions with location and annotations
	$\vert$	$loc\ annots\ texpr\_aux$	[V]	
$terminals$	::=			
	$\vert$	λ		
	$\vert$	$\longrightarrow$		
	$\vert$	$\rightarrow$		
	$\vert$	$\leadsto$		
	$\vert$	$\Rightarrow$		
	$\vert$	$\Leftarrow$		
	$\vert$	$\vdash$		
	$\vert$	$\in$		
	$\vert$	Π		
	$\vert$	$\forall$		
	$\vert$	$\multimap$		
	$\vert$	$\supset$		
	$\vert$	Σ		
	$\vert$	$\exists$		
	$\vert$	$\star$		
	$\vert$	$\times$		
	$\vert$	$\wedge$		
	$\vert$	$\bigwedge$		
	$\vert$	$\neg$		

	$\mid$	$=$		
	$\mid$	$\neq$		
	$\mid$	$\leq$		
	$\mid$	$\geq$		
	$\mid$	$\&$		
	$\mid$	$\cdot$		
	$\mid$	$\mid$		
	$\mid$	$+_{\mathrm{ptr}}$		
	$\mid$	$\mapsto$		
	$\mid$	$*$		
z	$::=$			OCaml arbitrary-width integer
	$\mid$	$\mathrm{of\_mem\_int}(\mathit{mem\_int})$	M	
	$\mid$	$\mathrm{of\_nat}(\mathit{nat})$	M	
	$\mid$	$\mathrm{of\_ctype}(\tau)$	M	size of a C type
	$\mid$	`ptr_size`	M	size of a pointer
$\mathbb{Q}$	$::=$			OCaml type for rational numbers
	$\mid$	$\frac{k_1}{k_2}$		
lit	$::=$			
	$\mid$	ident		
	$\mid$	$()$		
	$\mid$	bool		
	$\mid$	`int` z		
	$\mid$	$\mathbb{Q}$		
	$\mid$	`ptr` z		
$\mathit{bool\_op}$	$::=$			
	$\mid$	$\neg\, \mathit{term}$		

	$\mid$	$term_1 = term_2$
	$\mid$	$\bigwedge(\, \overline{term_i}^{\,i}\,)$
$arith\_op$	$::=$	
	$\mid$	$term_1 \times term_2$
$list\_op$	$::=$	
	$\mid$	$\texttt{nil}$
	$\mid$	$term_1 :: term_2$
	$\mid$	$[term_1, \,..\,, term_n]$
	$\mid$	$term^{(k)}$
$tuple\_op$	$::=$	
	$\mid$	$(term_1, \,..\,, term_n)$
	$\mid$	$term^{(k)}$
$pointer\_op$	$::=$	
	$\mid$	$\texttt{nullop}$
	$\mid$	$term_1 +_{\mathrm{ptr}} term_2$
$option\_op$	$::=$	
	$\mid$	$\texttt{none}\, BT\_t$
	$\mid$	$\texttt{some}\, term$
$array\_op$	$::=$	
	$\mid$	$term[\, \texttt{int}\, z]$
$param\_op$	$::=$	
	$\mid$	$term(term_1, \,..\,, term_n)$

$struct\_op$	$::=$			
	$\vert$	$term.member$		
ct_pred	$::=$			
	$\vert$	$\texttt{representable}\,(\tau, term)$		
	$\vert$	$\texttt{alignedI}\,(term_1, term_2)$		
$term\_aux$	$::=$			
	$\vert$	$arith\_op$		
	$\vert$	$bool\_op$		
	$\vert$	$tuple\_op$		
	$\vert$	$struct\_op$		
	$\vert$	$pointer\_op$		
	$\vert$	$list\_op$		
	$\vert$	$array\_op$		
	$\vert$	ct_pred		
	$\vert$	$option\_op$		
	$\vert$	$param\_op$		
$term$	$::=$			
	$\vert$	lit		
	$\vert$	$term\_aux\; bt$		
	$\vert$	$(term)$	S	parentheses
	$\vert$	$[term_1/ident]term_2$	M	
$point$	$::=$			points-to predicate
	$\vert$	$IT\_t_1 \mapsto_{z, IT\_t_2} IT\_t_3$		
$predicate\_name$	$::=$			names of predicates
	$\vert$	$Sctypes\_t$		

	$\vert$	$string$	
$predicate$	$::=$		arbitrary predicate
	$\vert$	$predicate\_name(IT\_t_1, IT\_t\_list_1 \mapsto IT\_t\_list_2)$	
$resource$	$::=$		
	$\vert$	$point$	
	$\vert$	$predicate$	
arg	$::=$		argument types
	$\vert$	$\Pi\, ident : \beta.arg$	
	$\vert$	$\forall\, ident : \beta.arg$	
	$\vert$	$resource \multimap arg$	
	$\vert$	$term \supset arg$	
	$\vert$	$\mathtt{I}$	
ret, _	$::=$		return types
	$\vert$	$\Sigma ident{:}\beta.\ ret$	
	$\vert$	$\exists ident{:}\beta.\ ret$	
	$\vert$	$resource \star ret$	
	$\vert$	$term \wedge ret$	
	$\vert$	$\mathtt{I}$	
$\mathcal{C}$	$::=$		computational var env
	$\vert$	$\cdot$	
	$\vert$	$\mathcal{C}, ident : BT\_t$	
	$\vert$	$\mathcal{C}, \mathcal{C}'$	
	$\vert$	$\mathrm{fresh}(\mathcal{C})$	M identical context except with fresh variable names
$\mathcal{L}$	$::=$		logical var env

	$\mid$	$\cdot$	
	$\mid$	$\mathcal{L}, \mathit{ident}$	
Φ	::=		constraints env
	$\mid$	$\cdot$	
	$\mid$	Φ, term	
	$\mid$	Φ, ret	temporary hack
$\mathcal{R}$	::=		resources env
	$\mid$	$\cdot$	
	$\mid$	$\mathcal{R}, \mathit{resource}$	
$\mathit{formula}$	::=		
	$\mid$	$\mathit{judgement}$	
	$\mid$	$\mathtt{smt}\,(\Phi \Rightarrow \mathit{ret})$	theorem to be proved: *ret* only consists of logical constraints
	$\mid$	$\mathit{ident} : \beta \in \mathcal{C}$	
	$\mid$	$\mathit{ident} : \mathtt{struct}\,\mathit{tag}\ \&\,\overline{\mathit{member}_i : \tau_i}^{\,i} \in \mathtt{Globals}$	
	$\mid$	$\overline{\mathcal{C}_i; \mathcal{L}_i; \Phi_i \vdash \mathit{mem\_val}_i \Rightarrow \mathtt{mem}\,y_i, \beta_i, \mathit{term}_i}^{\,i}$	
	$\mid$	$\overline{\mathcal{C}_i; \mathcal{L}_i; \Phi_i \vdash \mathit{value}_i \Rightarrow y_i, \beta_i, \mathit{term}_i}^{\,i}$	
	$\mid$	$\overline{\mathcal{C}_i; \mathcal{L}_i; \Phi_i \vdash \mathit{pexpr\_aux}_i \Rightarrow \mathit{ret}_i}^{\,i}$	
	$\mid$	$\overline{\mathit{pattern}_i : \beta_i \rightsquigarrow \mathcal{C}_i}^{\,i}$	
	$\mid$	$\overline{\mathcal{C}_i; \mathcal{L}_i; \Phi_i \vdash \mathit{tpexpr}_i \Leftarrow \mathit{ret}_i}^{\,i}$	
$\mathit{mem\_value\_jtype}$	::=		
	$\mid$	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{mem\_val} \Rightarrow \mathtt{mem}\,y, \beta, \mathit{term}$	
$\mathit{value\_jtype}$	::=		
	$\mid$	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{object\_value} \Rightarrow \mathtt{obj}\,\mathit{ident}, \beta, \mathit{term}$	
	$\mid$	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{value} \Rightarrow \mathit{ident}, \beta, \mathit{term}$	

$$
\begin{array}{lcl}
\textit{pexpr\_jtype} & ::= & \\
 & | & \mathcal{C}; \mathcal{L}; \Phi \vdash \textit{pexpr\_aux} \Rightarrow \textit{ret} \\
 & & \\
\textit{pattern\_jtype} & ::= & \\
 & | & \textit{pattern} : \beta \rightsquigarrow \mathcal{C} \\
 & | & \textit{ident\_or\_pattern} : \beta \rightsquigarrow \mathcal{C} \\
 & & \\
\textit{tpexpr\_jtype} & ::= & \\
 & | & \mathcal{C}; \mathcal{L}; \Phi \vdash \textit{pexpr} \Rightarrow \textit{ret} \\
 & | & \mathcal{C}; \mathcal{L}; \Phi \vdash \textit{tpexpr} \Leftarrow \textit{ret} \\
 & | & \mathcal{C}; \mathcal{L}; \Phi \vdash \textit{tpexpr\_aux} \Leftarrow \textit{ret} \\
 & & \\
\textit{expr\_jtype} & ::= & \\
 & | & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \textit{action\_aux} \Rightarrow \textit{ret} \\
 & | & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \textit{expr\_aux} \Rightarrow \textit{ret} \\
 & & \\
\textit{judgement} & ::= & \\
 & | & \textit{mem\_value\_jtype} \\
 & | & \textit{value\_jtype} \\
 & | & \textit{pexpr\_jtype} \\
 & | & \textit{pattern\_jtype} \\
 & | & \textit{tpexpr\_jtype} \\
 & | & \textit{expr\_jtype} \\
 & & \\
\textit{user\_syntax} & ::= & \\
 & | & \textit{ident} \\
 & | & \textit{impl\_const} \\
 & | & \textit{mem\_int} \\
 & | & \textit{member} \\
 & | & \textit{annots}
\end{array}
$$

$|\quad nat$
$|\quad n$
$|\quad loc$
$|\quad mem\_iv\_c$
$|\quad UB\_name$
$|\quad string$
$|\quad tyvar\_TY$
$|$
$|\quad mem\_order$
$|\quad linux\_mem\_order$
$|\quad k$
$|\quad bt$
$|\quad Sctypes\_t$
$|\quad tag$
$|\quad \beta$
$|\quad binop$
$|\quad polarity$
$|\quad ident$
$|\quad ident$
$|\quad name$
$|\quad mem\_ptr$
$|\quad mem\_val$
$|\quad \tau$
$|\quad object\_value$
$|\quad loaded\_value$
$|\quad \beta$
$|\quad value$
$|\quad ctor$
$|\quad ident\_opt\_\beta$

$$
\begin{array}{ll}
| & pattern\_aux \\
| & pattern \\
| & ident\_or\_pattern \\
| & ident \\
| & \tau \\
| & pexpr\_aux \\
| & pexpr \\
| & tpexpr\_aux \\
| & tpexpr \\
| & m\_kill\_kind \\
| & bool \\
| & action\_aux \\
| & action \\
| & memop \\
| & paction \\
| & expr\_aux \\
| & expr \\
| & texpr\_aux \\
| & texpr \\
| & terminals \\
| & z \\
| & \mathbb{Q} \\
| & lit \\
| & bool\_op \\
| & arith\_op \\
| & list\_op \\
| & tuple\_op \\
| & pointer\_op \\
| & BT\_t
\end{array}
$$

$|$ *option_op*
$|$ *array_op*
$|$ *param_op*
$|$ *struct_op*
$|$ *ct_pred*
$|$ *term_aux*
$|$ *term*
$|$ *IT_t*
$|$ *IT_t_list*
$|$ *point*
$|$ *predicate_name*
$|$ *predicate*
$|$ *resource*
$|$ *arg*
$|$ *ret*
$|$ $\mathcal{C}$
$|$ $\mathcal{L}$
$|$ Φ
$|$ $\mathcal{R}$
$|$ *formula*

$\boxed{\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{mem\_val} \Rightarrow \texttt{mem}\, y, \beta, \mathit{term}}$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{int}\, \mathit{mem\_int} \Rightarrow \texttt{mem}\, y, \texttt{integer}, y = \texttt{int}\, \mathrm{of\_mem\_int}(\mathit{mem\_int})} \quad \text{VAL\_OBJ\_MEM\_INT}$$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{nullptr} \Rightarrow \texttt{mem}\, y, \texttt{loc}, y = \texttt{nullop}} \quad \text{VAL\_OBJ\_MEM\_PTR\_NULL}$$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{funcptr}\, \mathit{ident} \Rightarrow \texttt{mem}\, y, \texttt{loc}, y = \mathit{ident}} \quad \text{VAL\_OBJ\_MEM\_PTR\_FUNC}$$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{concptr}\, \mathit{nat} \Rightarrow \texttt{mem}\, y, \texttt{loc}, y = \texttt{ptr}\, \mathrm{of\_nat}(\mathit{nat})} \quad \text{VAL\_OBJ\_MEM\_PTR\_CONC}$$

$$\frac{\overline{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{mem\_val}_i \Rightarrow \texttt{mem}\, y_i, \beta, \mathit{term}_i}^{\,i}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{array}\,(\,\overline{\mathit{mem\_val}_i}^{\,i}\,) \Rightarrow \texttt{mem}\, y, \texttt{array}\,\beta, \bigwedge(\,\overline{[y[\,\texttt{int}\, z_i]\,/y_i]\mathit{term}_i}^{\,i}\,)} \quad \textsc{Val\_Obj\_Mem\_Arr}$$

$$\frac{\overline{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{mem\_val}_i \Rightarrow \texttt{mem}\, y_i, \beta_i, \mathit{term}_i}^{\,i}}{\mathcal{C};\mathcal{L};\Phi \vdash (\,\texttt{struct}\, \mathit{tag})\{\,\overline{.\mathit{member}_i = \mathit{mem\_val}_i}^{\,i}\,\} \Rightarrow \texttt{mem}\, y, \texttt{struct}\, \mathit{tag}, \bigwedge(\,\overline{[y.\mathit{member}_i\,/y_i]\mathit{term}_i}^{\,i}\,)} \quad \textsc{Val\_Obj\_Mem\_Struct}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{object\_value} \Rightarrow \texttt{obj}\, \mathit{ident}, \beta, \mathit{term}}$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{mem\_int} \Rightarrow \texttt{obj}\, y, \texttt{integer}, y = \texttt{int}\,\mathrm{of\_mem\_int}(\mathit{mem\_int})} \quad \textsc{Val\_Obj\_Int}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{nullptr} \Rightarrow \texttt{obj}\, y, \texttt{loc}, y = \texttt{nullop}} \quad \textsc{Val\_Obj\_Ptr\_Null}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{funcptr}\, \mathit{ident} \Rightarrow \texttt{obj}\, y, \texttt{loc}, y = \mathit{ident}} \quad \textsc{Val\_Obj\_Ptr\_Func}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{concptr}\, \mathit{nat} \Rightarrow \texttt{obj}\, y, \texttt{loc}, y = \texttt{ptr}\,\mathrm{of\_nat}(\mathit{nat})} \quad \textsc{Val\_Obj\_Ptr\_Conc}$$

$$\frac{\overline{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{loaded\_value}_i \Rightarrow y_i, \beta, \mathit{term}_i}^{\,i}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{array}\,(\,\overline{\mathit{loaded\_value}_i}^{\,i}\,) \Rightarrow \texttt{obj}\, y, \texttt{array}\,\beta, \bigwedge(\,\overline{[y[\,\texttt{int}\, z_i]\,/y_i]\mathit{term}_i}^{\,i}\,)} \quad \textsc{Val\_Obj\_Arr}$$

$$\frac{\begin{array}{l}\mathit{ident} : \texttt{struct}\, \mathit{tag}\ \&\ \overline{\mathit{member}_i : \tau_i}^{\,i} \in \texttt{Globals}\\ \overline{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{mem\_val}_i \Rightarrow \texttt{mem}\, y_i, \beta_i, \mathit{term}_i}^{\,i}\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash (\,\texttt{struct}\, \mathit{tag})\{\,\overline{.\mathit{member}_i : \tau_i = \mathit{mem\_val}_i}^{\,i}\,\} \Rightarrow \texttt{obj}\, y, \texttt{struct}\, \mathit{tag}, \bigwedge(\,\overline{[y.\mathit{member}_i\,/y_i]\mathit{term}_i}^{\,i}\,)} \quad \textsc{Val\_Obj\_Struct}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{value} \Rightarrow \mathit{ident}, \beta, \mathit{term}}$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{object\_value} \Rightarrow \texttt{obj}\, y, \beta, \mathit{term}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{object\_value} \Rightarrow y, \beta, \mathit{term}} \quad \text{VAL\_OBJ}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{object\_value} \Rightarrow \texttt{obj}\, y, \beta, \mathit{term}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{specified}\, \mathit{object\_value} \Rightarrow y, \beta, \mathit{term}} \quad \text{VAL\_LOADED}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{Unit} \Rightarrow y, \texttt{unit}, y = ()} \quad \text{VAL\_UNIT}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{True} \Rightarrow y, \texttt{bool}, y = \texttt{true}} \quad \text{VAL\_TRUE}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{False} \Rightarrow y, \texttt{bool}, y = \texttt{false}} \quad \text{VAL\_FALSE}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{value}_1 \Rightarrow y_1, \beta, \mathit{term}_1 \,..\, \mathcal{C};\mathcal{L};\Phi \vdash \mathit{value}_n \Rightarrow y_n, \beta, \mathit{term}_n}{\mathcal{C};\mathcal{L};\Phi \vdash \beta[\mathit{value}_1, \,..\,, \mathit{value}_n] \Rightarrow y, [\beta], \bigwedge([y^{(k_1)}/y_1]\mathit{term}_1, \,..\,, [y^{(k_n)}/y_n]\mathit{term}_n)} \quad \text{VAL\_LIST}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{value}_1 \Rightarrow y_1, \beta_1, \mathit{term}_1 \,..\, \mathcal{C};\mathcal{L};\Phi \vdash \mathit{value}_n \Rightarrow y_n, \beta_n, \mathit{term}_n}{\mathcal{C};\mathcal{L};\Phi \vdash (\mathit{value}_1, \,..\,, \mathit{value}_n) \Rightarrow y, (\beta_1, \,..\,, \beta_n), \bigwedge([y^{(k_1)}/y_1]\mathit{term}_1, \,..\,, [y^{(k_n)}/y_n]\mathit{term}_n)} \quad \text{VAL\_TUPLE}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{pexpr\_aux} \Rightarrow \mathit{ret}}$

$$\frac{x : \beta \in \mathcal{C}}{\mathcal{C};\mathcal{L};\Phi \vdash x \Rightarrow \Sigma y{:}\beta.\ y = x \wedge \texttt{I}} \quad \text{PEXPR\_AUX\_VAR}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{value} \Rightarrow y, \beta, \mathit{term}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{value} \Rightarrow \Sigma y{:}\beta.\ \mathit{term} \wedge \texttt{I}} \quad \text{PEXPR\_AUX\_VAL}$$

$$\frac{\mathtt{smt}\,(\Phi \Rightarrow \mathtt{false} \wedge \mathtt{I})}{\mathcal{C};\mathcal{L};\Phi \vdash \mathtt{error}\,(\mathit{string}, v) \Rightarrow \mathit{ret}} \quad \textsc{PExpr\_Aux\_Error}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \mathtt{Nil}\,\beta(\,) \Rightarrow \Sigma y{:}[\beta].\ y = \mathtt{nil}\ \wedge \mathtt{I}} \quad \textsc{PExpr\_Aux\_Ctor\_Nil}$$

$$\frac{\begin{array}{c}\mathcal{C};\mathcal{L};\Phi \vdash v_1 \Rightarrow \Sigma\_{:}\beta.\ \_\\ \mathcal{C};\mathcal{L};\Phi \vdash v_2 \Rightarrow \Sigma\_{:}[\beta].\ \_\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathtt{Cons}(v_1, v_2) \Rightarrow \Sigma y{:}[\beta].\ y = v_1 :: v_2\ \wedge \mathtt{I}} \quad \textsc{PExpr\_Aux\_Ctor\_Cons}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash v_1 \Rightarrow \Sigma\_{:}\beta_1.\ \_ \,..\, \mathcal{C};\mathcal{L};\Phi \vdash v_n \Rightarrow \Sigma\_{:}\beta_n.\ \_}{\mathcal{C};\mathcal{L};\Phi \vdash \mathtt{Tuple}(v_1, .., v_n) \Rightarrow \Sigma y{:}(\beta_1, .., \beta_n).\ y = (v_1, .., v_n)\ \wedge \mathtt{I}} \quad \textsc{PExpr\_Aux\_Ctor\_Tuple}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash v_1 \Rightarrow \Sigma\_{:}\beta.\ \_ \,..\, \mathcal{C};\mathcal{L};\Phi \vdash v_n \Rightarrow \Sigma\_{:}\beta.\ \_}{\mathcal{C};\mathcal{L};\Phi \vdash \mathtt{Array}(v_1, .., v_n) \Rightarrow \Sigma y{:}\mathtt{array}\,\beta.\ \bigwedge(y[\,\mathtt{int}\,z_1] = v_1, .., y[\,\mathtt{int}\,z_n] = v_n)\ \wedge \mathtt{I}} \quad \textsc{PExpr\_Aux\_Ctor\_Array}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash v \Rightarrow \Sigma\_{:}\beta.\ \_}{\mathcal{C};\mathcal{L};\Phi \vdash \mathtt{Specified}(v) \Rightarrow \Sigma y{:}\beta.\ y = v\ \wedge \mathtt{I}} \quad \textsc{PExpr\_Aux\_Ctor\_Specified}$$

$$\frac{\begin{array}{c}\mathcal{C};\mathcal{L};\Phi \vdash v_1 \Rightarrow \Sigma\_{:}\mathtt{loc}.\ \_\\ \mathcal{C};\mathcal{L};\Phi \vdash v_2 \Rightarrow \Sigma\_{:}\mathtt{integer}.\ \_\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathtt{array\_shift}\,(v_1, \tau, v_2) \Rightarrow \Sigma y{:}\mathtt{loc}.\ y = v_1 +_{\mathrm{ptr}} v_2 \times \mathtt{int}\,\mathrm{of\_ctype}(\tau)\ \wedge \mathtt{I}} \quad \textsc{PExpr\_Aux\_Array\_Shift}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \mathtt{not}\,(v) \Rightarrow \Sigma y{:}\mathtt{bool}.\ y = (\neg\, v)\ \wedge \mathtt{I}} \quad \textsc{PExpr\_Aux\_Not}$$

$\boxed{\mathit{pattern} : \beta \rightsquigarrow \mathcal{C}}$

$\boxed{\mathit{ident\_or\_pattern} : \beta \rightsquigarrow \mathcal{C}}$

$\boxed{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{pexpr} \Rightarrow \mathit{ret}}$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{pexpr\_aux} \Rightarrow \mathit{ret}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{loc}\ \mathit{annots}\ \mathit{tyvar\_TY}\ \mathit{pexpr\_aux} \Rightarrow \mathit{ret}} \quad \text{PE}\textsc{xpr\_Aux}$$

$$\boxed{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{tpexpr} \Leftarrow \mathit{ret}}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{tpexpr\_aux} \Leftarrow \mathit{ret}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{loc}\ \mathit{annots}\ \mathit{tyvar\_TY}\ \mathit{tpexpr\_aux} \Leftarrow \mathit{ret}} \quad \text{TPE}\textsc{xpr\_Aux}$$

$$\boxed{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{tpexpr\_aux} \Leftarrow \mathit{ret}}$$

$$\frac{\texttt{smt}\,(\Phi \Rightarrow \texttt{false} \wedge \texttt{I})}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{undef}\,\mathit{loc}\ \mathit{UB\_name} \Leftarrow \mathit{ret}} \quad \text{TPE}\textsc{xpr\_Aux\_Undef}$$

$$\frac{\begin{array}{l}\mathcal{C};\mathcal{L};\Phi \vdash v \Rightarrow \Sigma\_{:}\texttt{bool}.\ \_\\ \mathcal{C};\mathcal{L};\Phi, v = \texttt{true} \vdash \mathit{tpexpr}_1 \Leftarrow \mathit{ret}\\ \mathcal{C};\mathcal{L};\Phi, v = \texttt{false} \vdash \mathit{tpexpr}_2 \Leftarrow \mathit{ret}\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{if}\ v\ \texttt{then}\,\mathit{tpexpr}_1\,\texttt{else}\,\mathit{tpexpr}_2 \Leftarrow \mathit{ret}} \quad \text{TPE}\textsc{xpr\_Aux\_If}$$

$$\frac{\begin{array}{l}\mathcal{C};\mathcal{L};\Phi \vdash v \Rightarrow \Sigma\_{:}\beta.\ \mathit{ret}'\\ \texttt{smt}\,(\Phi, \mathit{ret}' \Rightarrow \mathit{ret})\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{done}\ v \Leftarrow \Sigma y{:}\beta.\ \mathit{ret}} \quad \text{TPE}\textsc{xpr\_Aux\_Done}$$

$$\frac{\begin{array}{l}\mathcal{C};\mathcal{L};\Phi \vdash \mathit{pexpr} \Rightarrow \Sigma y{:}\beta.\ \mathit{ret}'\\ \mathit{ident\_or\_pattern} : \beta \rightsquigarrow \mathcal{C}'\\ \mathcal{C}, \mathrm{fresh}(\mathcal{C}');\mathcal{L};\Phi \vdash [\mathrm{fresh}(\mathcal{C}')/\mathcal{C}']\mathit{tpexpr} \Leftarrow \mathit{ret}\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{let}\,\mathit{ident\_or\_pattern} = \mathit{pexpr}\,\texttt{in}\,\mathit{tpexpr} \Leftarrow \mathit{ret}} \quad \text{TPE}\textsc{xpr\_Aux\_Let}$$

$$\frac{\begin{array}{c}\mathcal{C};\mathcal{L};\Phi \vdash v \Rightarrow \Sigma\_{:}\beta.\ \_\\ \overline{\mathit{pattern}_i : \beta \rightsquigarrow \mathcal{C}_i}^{\,i}\\ \overline{\mathcal{C}, \mathrm{fresh}(\mathcal{C}_i);\mathcal{L};\Phi \vdash [\mathrm{fresh}(\mathcal{C}_i)/\mathcal{C}_i]\mathit{tpexpr}_i \Leftarrow \mathit{ret}}^{\,i}\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{case}\ v\ \texttt{of}\ \overline{\mid \mathit{pattern}_i \Rightarrow \mathit{tpexpr}_i}^{\,i}\ \texttt{end} \Leftarrow \mathit{ret}}\quad \textsc{TPExpr\_Aux\_Case}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{action\_aux} \Rightarrow \mathit{ret}}$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash v \Rightarrow \Sigma\_{:}\texttt{integer}.\ \_}{\mathcal{C};\mathcal{L};\Phi;\cdot \vdash \texttt{create}\,(v,\tau) \Rightarrow \Sigma y_p{:}\texttt{loc}.\ \exists v'{:}\mathrm{of\_ctype}(\tau).\ \texttt{representable}\,(\tau*, y_p)\ \wedge \texttt{alignedI}\,(v, y_p)\ \wedge \tau(y_p, [\,] \mapsto [v', \texttt{false}]) \star \texttt{I}}\quad \textsc{Action\_Create}$$

$$\frac{\begin{array}{l}\mathcal{C};\mathcal{L};\Phi \vdash v_p \Rightarrow \Sigma\_{:}\texttt{loc}.\ \_\\ \mathcal{C};\mathcal{L};\Phi \vdash v_2 \Rightarrow \Sigma\_{:}\mathrm{of\_ctype}(\tau).\ \_\\ \texttt{smt}\,(\Phi \Rightarrow \texttt{representable}\,(\tau, v_2)\ \wedge \texttt{I})\end{array}}{\mathcal{C};\mathcal{L};\Phi;\cdot,\tau(v_p,[\,]\mapsto[v_1,\_]) \vdash \texttt{store}\,(\_,\tau,v_p,v_2,\_) \Rightarrow \Sigma\_{:}\texttt{unit}.\ \tau(v_p,[\,]\mapsto[v_2,\texttt{true}]) \star \texttt{I}}\quad \textsc{Action\_Store}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash v_p \Rightarrow \Sigma\_{:}\texttt{loc}.\ \_}{\mathcal{C};\mathcal{L};\Phi;\cdot,\tau(v_p,[\,]\mapsto[v_1,\_]) \vdash \texttt{kill}\,(\texttt{static}\,\tau, v_p) \Rightarrow \Sigma\_{:}\texttt{unit}.\ \texttt{I}}\quad \textsc{Action\_Kill\_Static}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{expr\_aux} \Rightarrow \mathit{ret}}$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{pexpr} \Rightarrow \mathit{ret}}{\mathcal{C};\mathcal{L};\Phi;\cdot \vdash \texttt{pure}\,(\mathit{pexpr}) \Rightarrow \mathit{ret}}\quad \textsc{Expr\_Aux\_Pure}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{action\_aux} \Rightarrow \mathit{ret}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{loc}\,\mathit{action\_aux} \Rightarrow \mathit{ret}}\quad \textsc{Expr\_Aux\_Action}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{action\_aux} \Rightarrow \mathit{ret}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \neg\,(\mathit{loc}\,\mathit{action\_aux}) \Rightarrow \mathit{ret}}\quad \textsc{Expr\_Aux\_Neg\_Action}$$

```
Definition rules:        42 good    0 bad
Definition rule clauses: 83 good    0 bad
```