ident, x, y, y_p, y_f, _, *abbrev*	subscripts: p for pointers, f for functions
n, i, j	index variables
impl_const	implementation-defined constant
mem_int	memory integer value
member	C struct/union member name
	Ott-hack, ignore (annotations)
nat	OCaml arbitrary-width natural number
mem_ptr	abstract pointer value
mem_val	abstract memory value
	Ott-hack, ignore (locations)
mem_iv_c	OCaml type for memory constraints on integer values
UB_name	undefined behaviour
string	OCaml string
	Ott-hack, ignore (OCaml type variable TY)
	Ott-hack, ignore (Symbol.prefix)
mem_order, _	OCaml type for memory order
linux_mem_order	OCaml type for Linux memory order
	Ott-hack, ignore (OCaml type variable bt)
logical_val	logical values (to be specified)

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
	|	$\overline{\beta_i}^{\,i}$		tuple
	|	`struct` *tag*		struct
	|	$\{\beta\}$		set
	|	`opt` (β)		option
	|	$\beta \rightarrow \beta'$		parameter types
	|	β_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

	`>`	greater than
	`<`	less than
	`>=`	greater than or equal to
	`<=`	less than or equal to
	`/\`	conjucntion
	`\/`	disjunction

$binop_{arith}$	::=		arithmentic binary operators
	|	`+`	
	|	`-`	
	|	`*`	
	|	`/`	
	|	`rem_t`	
	|	`rem_f`	
	|	`^`	

$binop_{rel}$	::=		relational binary operators
	|	`=`	
	|	`>`	
	|	`<`	
	|	`>=`	
	|	`<=`	

$binop_{bool}$	::=		boolean binary operators
	|	`/\`	
	|	`\/`	

object_value	::=		C object values (inhabitants of object types), which can be read/stored
	|	*mem_int*	integer value
	|	*mem_ptr*	pointer value

	$\mid$	$\texttt{array}\,(\overline{loaded\_value_i}^{\,i})$	C array value
	$\mid$	$(\texttt{struct}\,ident)\{\,\overline{.member_i{:}\tau_i = mem\_val_i}^{\,i}\,\}$	C struct value
	$\mid$	$(\texttt{union}\,ident)\{\,.member = mem\_val\}$	C union value
loaded_value	::=		potentially unspecified C object values
	$\mid$	$\texttt{specified}\,object\_value$	specified loaded value
value	::=		Core values
	$\mid$	$object\_value$	C object value
	$\mid$	$loaded\_value$	loaded C object value
	$\mid$	`Unit`	unit
	$\mid$	`True`	boolean true
	$\mid$	`False`	boolean false
	$\mid$	$\beta[\overline{value_i}^{\,i}]$	list
	$\mid$	$(\overline{value_i}^{\,i})$	tuple
ctor_val	::=		data constructors
	$\mid$	$\texttt{Nil}\,\beta$	empty list
	$\mid$	`Cons`	list cons
	$\mid$	`Tuple`	tuple
	$\mid$	`Array`	C array
	$\mid$	`Specified`	non-unspecified loaded value
ctor_expr	::=		data constructors
	$\mid$	`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

	`IvOR`	bitwise OR
	`IvXOR`	bitwise XOR
	`Fvfromint`	cast integer to floating value
	`Ivfromfloat`	cast floating to integer value

name	::=	
	ident	Core identifier
	impl_const	implementation-defined constant

pval	::=	pure values
	ident	Core identifier
	impl_const	implementation-defined constant
	value	Core values
	`constrained` $(\overline{mem\_iv\_c_i, pval_i}^{\,i})$	constrained value
	`error` $(string, pval)$	impl-defined static error
	$ctor\_val(\overline{pval_i}^{\,i})$	data constructor application
	$(\texttt{struct}\, ident)\{\overline{.member_i = pval_i}^{\,i}\}$	C struct expression
	$(\texttt{union}\, ident)\{.member = pval\}$	C union expression

pexpr	::=	pure expressions
	pval	pure values
	$ctor\_expr(\overline{pval_i}^{\,i})$	data constructor application
	`array_shift` $(pval_1, \tau, pval_2)$	pointer array shift
	`member_shift` $(pval, ident, member)$	pointer struct/union member shift
	`not` $(pval)$	boolean not
	$pval_1\ binop\ pval_2$	binary operations
	`memberof` $(ident, member, pval)$	C struct/union member access
	$name(\overline{pval_i}^{\,i})$	pure function call
	`assert_undef` $(pval,\ UB\_name)$	
	`bool_to_integer` $(pval)$	

	|	$\texttt{conv\_int}\,(\tau, pval)$		
	|	$\texttt{wrapI}\,(\tau, pval)$		
tpval	::=			top-level pure values
	|	$\texttt{undef}\ UB\_name$		undefined behaviour
	|	$\texttt{done}\,pval$		pure done
ident_opt_β	::=			type annotated optional identifier
	|	$\_{:}\beta$		
	|	$ident{:}\beta$		
pattern	::=			
	|	$ident\_opt\_\beta$		
	|	$ctor\_val(\overline{pattern_i}^{\,i})$		
ident_or_pattern	::=			
	|	$ident$		
	|	$pattern$		
tpexpr	::=			top-level pure expressions
	|	$tpval$		top-level pure values
	|	$\texttt{case}\,pval\,\texttt{of}\,\overline{\mid pattern_i \Rightarrow tpexpr_i}^{\,i}\,\texttt{end}$		pattern matching
	|	$\texttt{let}\,ident\_or\_pattern = pexpr\,\texttt{in}\,tpexpr$		pure let
	|	$\texttt{if}\,pval\,\texttt{then}\,tpexpr_1\,\texttt{else}\,tpexpr_2$		pure if
	|	$[\mathcal{C}/\mathcal{C}']tpexpr$	M	simul-sub all vars in $\mathcal{C}$ for all vars in $\mathcal{C}'$ in $tpexpr$
m_kill_kind	::=			
	|	$\texttt{dynamic}$		
	|	$\texttt{static}\,\tau$		

bool, _	::=		OCaml booleans
	|	`true`	
	|	`false`	
int, _	::=		OCaml fixed-width integer
	|	i	literal integer
mem_action	::=		memory actions
	|	`create` $(pval, \tau)$	
	|	`create_readonly` $(pval_1, \tau, pval_2)$	
	|	`alloc` $(pval_1, pval_2)$	
	|	`kill` $(m\_kill\_kind, pval)$	
	|	`store` $(bool, \tau, pval_1, pval_2, mem\_order)$	true means store is locking
	|	`load` $(\tau, pval, mem\_order)$	
	|	`rmw` $(\tau, pval_1, pval_2, pval_3, mem\_order_1, mem\_order_2)$	
	|	`fence` (mem_order)	
	|	`cmp_exch_strong` $(\tau, pval_1, pval_2, pval_3, mem\_order_1, mem\_order_2)$	
	|	`cmp_exch_weak` $(\tau, pval_1, pval_2, pval_3, mem\_order_1, mem\_order_2)$	
	|	`linux_fence` $(linux\_mem\_order)$	
	|	`linux_load` $(\tau, pval, linux\_mem\_order)$	
	|	`linux_store` $(\tau, pval_1, pval_2, linux\_mem\_order)$	
	|	`linux_rmw` $(\tau, pval_1, pval_2, linux\_mem\_order)$	
polarity	::=		polarities for memory actions
	|	`Pos`	sequenced by `let weak` and `let strong`
	|	`Neg`	only sequenced by `let strong`
pol_mem_action	::=		memory actions with polarity
	|	*polarity* *mem_action*	

mem_op	::=		operations involving the memory state
	|	$pval_1 \equiv pval_2$	pointer equality comparison
	|	$pval_1 \neq pval_2$	pointer inequality comparison
	|	$pval_1 < pval_2$	pointer less-than comparison
	|	$pval_1 > pval_2$	pointer greater-than comparison
	|	$pval_1 \leq pval_2$	pointer less-than comparison
	|	$pval_1 \geq pval_2$	pointer greater-than comparison
	|	$pval_1 -_\tau pval_2$	pointer subtraction
	|	`intFromPtr` $(\tau_1, \tau_2, pval)$	cast of pointer value to integer value
	|	`ptrFromInt` $(\tau_1, \tau_2, pval)$	cast of integer value to pointer value
	|	`ptrValidForDeref` $(\tau, pval)$	dereferencing validity predicate
	|	`ptrWellAligned` $(\tau, pval)$	
	|	`ptrArrayShift` $(pval_1, \tau, pval_2)$	
	|	`memcpy` $(pval_1, pval_2, pval_3)$	
	|	`memcmp` $(pval_1, pval_2, pval_3)$	
	|	`realloc` $(pval_1, pval_2, pval_3)$	
	|	`va_start` $(pval_1, pval_2)$	
	|	`va_copy` $(pval)$	
	|	`va_arg` $(pval, \tau)$	
	|	`va_end` $(pval)$	
tval	::=		(effectful) top-level values
	|	`done` $pval$	end of top-level expression
	|	`undef` UB_name	undefined behaviour
seq_expr	::=		sequential (effectful) expressions
	|	$pval$	pure values
	|	`ccall` $(\tau, pval, \overline{pval_i}^{\,i})$	C function call
	|	`pcall` $(name, \overline{pval_i}^{\,i})$	procedure call

seq_texpr	::=		sequential top-level (effectful) expressions
	|	*tval*	(effectful) top-level values
	|	`run` *ident* $pval_1, .., pval_n$	run from label
	|	`nd` $(pval_1, .., pval_n)$	nondeterministic choice
	|	`let` *ident_or_pattern* $=$ *seq_expr* `in` *texpr*	pure sequencing
	|	`case` *pval* `with` $\overline{\mid pattern_i \Rightarrow texpr_i}^{\,i}$ `end`	pattern matching
	|	`if` *pval* `then` $texpr_1$ `else` $texpr_2$	conditional
	|	`bound` $[int](is\_texpr)$	limit scope of indet seq behaviour, absent at runtime
is_expr	::=		indet seq (effectful) expressions
	|	`memop` (mem_op)	pointer op involving memory
	|	*pol_mem_action*	memory action
	|	`unseq` $(texpr_1, .., texpr_n)$	unsequenced expressions
is_texpr	::=		indet seq top-level (effectful) expressions
	|	`let weak` *pattern* $=$ *is_expr* `in mu_texpr_aux`	weak sequencing
	|	`let strong` *ident_or_pattern* $=$ *is_expr* `in mu_texpr_aux`	strong sequencing
texpr	::=		top-level (effectful) expressions
	|	*seq_texpr*	sequential (effectful) expressions
	|	*is_texpr*	indet seq (effectful) expressions
terminals	::=		
	|	λ	
	|	$\longrightarrow$	
	|	$\rightarrow$	
	|	$\rightsquigarrow$	
	|	$\Rightarrow$	
	|	$\Leftarrow$	
	|	$\vdash$	

| $\in$
| Π
| $\forall$
| $\multimap$
| $\supset$
| Σ
| $\exists$
| $\star$
| $\times$
| $\wedge$
| $\bigwedge$
| $\neg$
| $=$
| $\neq$
| $\leq$
| $\geq$
| $\&$
| $.$
| $|$
| $+_{\text{ptr}}$
| $\mapsto$
| $*$
| $::$
| $\checkmark$
| $:$
| $.$
| $.$
| $\gg$
| $::$

	$\mid$	$\hat{}$	
	$\mid$	$\bigvee$	
	$\mid$	$\equiv$	
z	::=		OCaml arbitrary-width integer
	$\mid$	1	M
	$\mid$	0	M
	$\mid$	i	M literal integer
	$\mid$	to_int(mem_int)	M
	$\mid$	size_of(τ)	M size of a C type
	$\mid$	offset_of$_{tag}$($member$)	M offset of a struct member
	$\mid$	`ptr_size`	M size of a pointer
	$\mid$	max_int$_\tau$	M maximum value of int of type τ
	$\mid$	min_int$_\tau$	M minimum value of int of type τ
$\mathbb{Q}$	::=		OCaml type for rational numbers
	$\mid$	$\frac{int_1}{int_2}$	
lit	::=		
	$\mid$	*ident*	
	$\mid$	`unit`	
	$\mid$	*bool*	
	$\mid$	z	
	$\mid$	$\mathbb{Q}$	
bool_op	::=		
	$\mid$	$\neg\, term$	
	$\mid$	$term_1 = term_2$	
	$\mid$	$\bigwedge(\overline{term_i}^{\,i})$	
	$\mid$	$\bigvee(\overline{term_i}^{\,i})$	

	$term_1\ binop_{bool}\ term_2$	M	
	$\texttt{if}\ term_1\ \texttt{then}\ term_2\ \texttt{else}\ term_3$		
arith_op	::=		
	$term_1 + term_2$		
	$term_1 - term_2$		
	$term_1 \times term_2$		
	$term_1/term_2$		
	$term_1\ \texttt{rem\_t}\ term_2$		
	$term_1\ \texttt{rem\_f}\ term_2$		
	$term_1\ \texttt{\^{}}\ term_2$		
	$term_1\ binop_{arith}\ term_2$	M	
cmp_op	::=		
	$term_1 < term_2$		less than
	$term_1 \leq term_2$		less than or equal
	$term_1\ binop_{rel}\ term_2$	M	
list_op	::=		
	$\texttt{nil}$		
	$\texttt{tl}\ term$		
	$term^{(int)}$		
tuple_op	::=		
	$(\overline{term_i}^{\,i})$		
	$term^{(int)}$		
pointer_op	::=		
	$\texttt{of\_mem\_ptr}\ mem\_ptr$		
	$term_1 +_{\mathrm{ptr}} term_2$		

$$
\begin{array}{lll}
option\_op & ::= & \\
 & | & \texttt{none}\ BT\_t \\
 & | & \texttt{some}\ term \\
 & & \\
array\_op & ::= & \\
 & | & term_1[term_2] \\
 & & \\
param\_op & ::= & \\
 & | & ident{:}\beta.\ term \\
 & | & term(term_1,\ ..,term_n) \\
 & & \\
struct\_op & ::= & \\
 & | & term.member \\
 & & \\
ct\_pred & ::= & \\
 & | & \texttt{representable}\ (\tau, term) \\
 & | & \texttt{alignedI}\ (term_1, term_2) \\
 & & \\
term,\ \_ & ::= & \\
 & | & lit \\
 & | & arith\_op \\
 & | & bool\_op \\
 & | & cmp\_op \\
 & | & tuple\_op \\
 & | & struct\_op \\
 & | & pointer\_op \\
 & | & list\_op \\
 & | & array\_op \\
 & | & ct\_pred \\
 & | & option\_op
\end{array}
$$

	$param\_op$		
	$(term)$	S	parentheses
	$[term_1/ident]term_2$	M	substitute $term_1$ for $ident$ in $term_2$
	$pval$	M	only the ones which can be embeded into the SMT value grammar, so no array literals
	$resource$		

$terms$	::=		non-empty list of terms
	$[term_1, \ldots, term_n]$		

$predicate\_name$	::=		names of predicates
	$Sctypes\_t$		C type
	$string$		arbitrary

$init$,	::=		initialisation status
	✓		initialised
	×		uninitalised

$predicate$	::=		arbitrary predicate
	$terms_1 \, \mathbb{Q}^{init}_{\mapsto \, predicate\_name} \, terms_2$		

$resource$	::=		
	$predicate$		

$spine\_elem$	::=		spine element
	$pval$		pure value
	$logical\_val$		logical variable
	$resource$		resource

arg	::=		argument types
	$\Pi \, ident{:}\beta. \, arg$		

	$\forall\, ident{:}\beta.\ arg$		
	$resource \multimap arg$		
	$term \supset arg$		
	ret		
	$[spine\_elem/ident]arg$	M	

$ret,\ \_$	::=		return types
	$\Sigma\, ident{:}\beta.\ ret$		
	$\exists\, ident{:}\beta.\ ret$		
	$resource \star ret$		
	$term \wedge ret$		
	$\mathtt{I}$		

$\mathcal{C}$	::=		computational var env
	.		
	$\mathcal{C}, ident{:}BT\_t$		
	$\overline{\mathcal{C}_i}^{\,i}$		
	fresh($\mathcal{C}$)	M	identical context except with fresh variable names

$\mathcal{L}$	::=		logical var env
	.		
	$\mathcal{L}, ident{:}BT\_t$		

Φ	::=		constraints env
	.		
	$\Phi, term$		

$\mathcal{R}$	::=		resources env
	.		
	$\mathcal{R}, resource$		

formula	::=		
	|	$\mathit{judgement}$	
	|	$\mathit{abbrev} \equiv \mathit{term}$	
	|	$\texttt{smt}\,(\Phi \Rightarrow \mathit{term})$	
	|	$\mathit{ident}{:}\beta \in \mathcal{C}$	
	|	$\mathit{ident}{:}\,\texttt{struct}\,\mathit{tag}\,\&\,\overline{\mathit{member}_i{:}\tau_i}^{\,i} \in \texttt{Globals}$	
	|	$\overline{\mathcal{C}_i; \mathcal{L}_i; \Phi_i \vdash \mathit{mem\_val}_i \Rightarrow \texttt{mem}\,y_i{:}\beta_i.\,\mathit{term}_i}^{\,i}$	dependent on memory object model
	|	$\overline{\mathcal{C}_i; \mathcal{L}_i; \Phi_i \vdash \mathit{pval}_i \Rightarrow \mathit{ident}_i{:}\beta_i.\,\mathit{term}_i}^{\,i}$	
	|	$\overline{\mathit{pattern}_i{:}\beta_i \rightsquigarrow \mathcal{C}_i}^{\,i}$	
	|	$\overline{\mathcal{C}_i; \mathcal{L}_i; \Phi_i \vdash \mathit{tpexpr}_i \Leftarrow y_i{:}\beta_i.\,\mathit{term}_i}^{\,i}$	
	|	$\mathcal{L} \vdash \mathit{logical\_val}{:}\beta$	
object_value_jtype	::=		
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{object\_value} \Rightarrow \texttt{obj}\,\mathit{ident}{:}\beta.\,\mathit{term}$	
pval_jtype	::=		
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pval} \Rightarrow \mathit{ident}{:}\beta.\,\mathit{term}$	
spine_jtype	::=		
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{\mathit{spine\_elem}_i}^{\,i} :: \mathit{arg} \gg \mathit{ret}$	
pexpr_jtype	::=		
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pexpr} \Rightarrow \mathit{ident}{:}\beta.\,\mathit{term}$	
pattern_jtype	::=		
	|	$\mathit{pattern}{:}\beta \rightsquigarrow \mathcal{C}$	
	|	$\mathit{ident\_or\_pattern}{:}\beta \rightsquigarrow \mathcal{C}$	
tpval_jtype	::=		
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{tpval} \Leftarrow \mathit{ident}{:}\beta.\,\mathit{term}$	

$$
\begin{array}{lll}
\mathit{tpexpr\_jtype} & ::= & \\
 & | & \mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{tpexpr} \Leftarrow \mathit{ident}{:}\beta.\ \mathit{term} \\
 & & \\
\mathit{action\_jtype} & ::= & \\
 & | & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{mem\_action} \Rightarrow \mathit{ret} \\
 & | & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{seq\_expr} \Rightarrow \mathit{ret} \\
 & | & \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{is\_expr} \Rightarrow \mathit{ret} \\
 & & \\
\mathit{judgement} & ::= & \\
 & | & \mathit{object\_value\_jtype} \\
 & | & \mathit{pval\_jtype} \\
 & | & \mathit{spine\_jtype} \\
 & | & \mathit{pexpr\_jtype} \\
 & | & \mathit{pattern\_jtype} \\
 & | & \mathit{tpval\_jtype} \\
 & | & \mathit{tpexpr\_jtype} \\
 & | & \mathit{action\_jtype} \\
 & & \\
\mathit{user\_syntax} & ::= & \\
 & | & \mathit{ident} \\
 & | & n \\
 & | & \mathit{impl\_const} \\
 & | & \mathit{mem\_int} \\
 & | & \mathit{member} \\
 & | & \\
 & | & \mathit{nat} \\
 & | & \mathit{mem\_ptr} \\
 & | & \mathit{mem\_val} \\
 & | & \\
 & | & \mathit{mem\_iv\_c}
\end{array}
$$

| UB_name
| $string$
|
|
| mem_order
| $linux\_mem\_order$
|
| $logical\_val$
| $Sctypes\_t$
| tag
| β
| $binop$
| $binop_{arith}$
| $binop_{rel}$
| $binop_{bool}$
| $ident$
| τ
| $ident$
| $object\_value$
| $loaded\_value$
| β
| $value$
| $ctor\_val$
| $ctor\_expr$
| τ
| $name$
| $pval$
| $pval$
| $pexpr$

| *pexpr*
| *tpval*
| *tpval*
| *ident_opt_β*
| *pattern*
| *pattern*
| *ident_or_pattern*
| *tpexpr*
| *tpexpr*
| *m_kill_kind*
| *bool*
| *int*
| *mem_action*
| *mem_action*
| *polarity*
| *pol_mem_action*
| *mem_op*
| *tval*
| *tval*
| *seq_expr*
| *seq_expr*
| *seq_texpr*
| *seq_texpr*
| *is_expr*
| *is_expr*
| *is_texpr*
| *is_texpr*
| *texpr*
| *terminals*

| z
| $\mathbb{Q}$
| lit
| $bool\_op$
| $arith\_op$
| cmp_op
| $list\_op$
| $tuple\_op$
| $pointer\_op$
| BT_t
| $option\_op$
| $array\_op$
| $param\_op$
| $struct\_op$
| ct_pred
| $term$
| $term$
| $term$
| $terms$
| $predicate\_name$
| $init$
| $predicate$
| $resource$
| $spine\_elem$
| arg
| ret
| $\mathcal{C}$
| $\mathcal{L}$
| Φ

| $\mathcal{R}$
| *formula*

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

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{mem\_int} \Rightarrow \texttt{obj}\ y{:}\texttt{integer}.\ y = \mathrm{to\_int}(\mathit{mem\_int})}\ \textsc{Pval\_Obj\_Int}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{mem\_ptr} \Rightarrow \texttt{obj}\ y{:}\texttt{loc}.\ y = \texttt{of\_mem\_ptr}\ \mathit{mem\_ptr}}\ \textsc{Pval\_Obj\_Ptr}$$

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

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

$\boxed{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{pval} \Rightarrow \mathit{ident}{:}\beta.\ \mathit{term}}$

$$\frac{x{:}\beta \in \mathcal{C}}{\mathcal{C};\mathcal{L};\Phi \vdash x \Rightarrow y{:}\beta.\ y = x}\ \textsc{Pval\_Var}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{object\_value} \Rightarrow \texttt{obj}\ y{:}\beta.\ \mathit{term}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{object\_value} \Rightarrow y{:}\beta.\ \mathit{term}}\ \textsc{Pval\_Obj}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{object\_value} \Rightarrow \texttt{obj}\ y{:}\beta.\ \mathit{term}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{specified}\,\mathit{object\_value} \Rightarrow y{:}\beta.\ \mathit{term}}\ \textsc{Pval\_Loaded}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{Unit} \Rightarrow y{:}\texttt{unit}.\ y = \texttt{unit}} \quad \textsc{Pval\_Unit}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{True} \Rightarrow y{:}\texttt{bool}.\ y = \texttt{true}} \quad \textsc{Pval\_True}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{False} \Rightarrow y{:}\texttt{bool}.\ y = \texttt{false}} \quad \textsc{Pval\_False}$$

$$\frac{\overline{\mathcal{C};\mathcal{L};\Phi \vdash value_i \Rightarrow y_i{:}\beta.\ term_i}^{\,i}}{\mathcal{C};\mathcal{L};\Phi \vdash \beta[\overline{value_i}^{\,i}] \Rightarrow y{:}[\beta].\ \bigwedge(\overline{[y^{(i)}/y_i]term_i}^{\,i})} \quad \textsc{Pval\_List}$$

$$\frac{\overline{\mathcal{C};\mathcal{L};\Phi \vdash value_i \Rightarrow y_i{:}\beta_i.\ term_i}^{\,i}}{\mathcal{C};\mathcal{L};\Phi \vdash (\overline{value_i}^{\,i}) \Rightarrow y{:}\overline{\beta_i}^{\,i}.\ \bigwedge(\overline{[y^{(i)}/y_i]term_i}^{\,i})} \quad \textsc{Pval\_Tuple}$$

$$\frac{\texttt{smt}\,(\Phi \Rightarrow \texttt{false})}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{error}\,(string, pval) \Rightarrow y{:}\beta.\ term} \quad \textsc{Pval\_Error}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{Nil}\,\beta(\,) \Rightarrow y{:}[\beta].\ y = \texttt{nil}} \quad \textsc{Pval\_Ctor\_Nil}$$

$$\frac{\begin{array}{c}\mathcal{C};\mathcal{L};\Phi \vdash pval_1 \Rightarrow y_1{:}\beta.\ term_1 \\ \mathcal{C};\mathcal{L};\Phi \vdash pval_2 \Rightarrow y_2{:}[\beta].\ term_2\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{Cons}(pval_1, pval_2) \Rightarrow y{:}[\beta].\ [y^{(1)}/y_1]term_1 \wedge [\texttt{tl}\ y/y_2]term_2} \quad \textsc{Pval\_Ctor\_Cons}$$

$$\frac{\overline{\mathcal{C};\mathcal{L};\Phi \vdash pval_i \Rightarrow y_i{:}\beta_i.\ term_i}^{\,i}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{Tuple}(\overline{pval_i}^{\,i}) \Rightarrow y{:}\overline{\beta_i}^{\,i}.\ \bigwedge(\overline{[y^{(i)}/y_i]term_i}^{\,i})} \quad \textsc{Pval\_Ctor\_Tuple}$$

$$\frac{\overline{\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pval}_i \Rightarrow y_i{:}\beta.\ \mathit{term}_i}^{\,i}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \mathtt{Array}(\overline{\mathit{pval}_i}^{\,i}) \Rightarrow y{:}\mathtt{array}\,\beta.\ \bigwedge(\overline{[y[i]/y_i]\mathit{term}_i}^{\,i})} \quad \textsc{Pval\_Ctor\_Array}$$

$$\frac{\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pval} \Rightarrow y{:}\beta.\ \mathit{term}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \mathtt{Specified}(\mathit{pval}) \Rightarrow y{:}\beta.\ \mathit{term}} \quad \textsc{Pval\_Ctor\_Specified}$$

$$\frac{\overline{\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pval}_i \Rightarrow y_i{:}\beta_i.\ \mathit{term}_i}^{\,i}}{\mathcal{C}; \mathcal{L}; \Phi \vdash (\,\mathtt{struct}\,\mathit{tag})\{\,\overline{.\mathit{member}_i = \mathit{pval}_i}^{\,i}\,\} \Rightarrow y{:}\mathtt{struct}\,\mathit{tag}.\ \bigwedge(\overline{[y.\mathit{member}_i/y_i]\mathit{term}_i}^{\,i})} \quad \textsc{Pval\_Struct}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{\mathit{spine\_elem}_i}^{\,i} :: \mathit{arg} \gg \mathit{ret}}$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash {::}\mathit{ret} \gg \mathit{ret}} \quad \textsc{Spine\_Empty}$$

$$\frac{\begin{array}{c}\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pval} \Rightarrow \_{:}\beta.\ \_ \\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{\mathit{spine\_elem}_i}^{\,i} :: [\mathit{pval}/x]\mathit{arg} \gg \mathit{ret}\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{pval},\ \overline{\mathit{spine\_elem}_i}^{\,i} :: \Pi\, x{:}\beta.\ \mathit{arg} \gg \mathit{ret}} \quad \textsc{Spine\_Computational}$$

$$\frac{\begin{array}{c}\mathcal{L} \vdash \mathit{logical\_val}{:}\beta \\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{\mathit{spine\_elem}_i}^{\,i} :: [\mathit{logical\_val}/x]\mathit{arg} \gg \mathit{ret}\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{logical\_val},\ \overline{\mathit{spine\_elem}_i}^{\,i} :: \forall\, x{:}\beta.\ \mathit{arg} \gg \mathit{ret}} \quad \textsc{Spine\_Logical}$$

$$\frac{\begin{array}{c}\mathtt{smt}\,(\Phi \Rightarrow \mathit{resource} = \mathit{resource}') \\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{\mathit{spine\_elem}_i}^{\,i} :: \mathit{arg} \gg \mathit{ret}\end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}, \mathit{resource} \vdash \mathit{resource}',\ \overline{\mathit{spine\_elem}_i}^{\,i} :: \mathit{resource}' \multimap \mathit{arg} \gg \mathit{ret}} \quad \textsc{Spine\_Resource}$$

$$\frac{\begin{array}{c}\texttt{smt}\,(\Phi \Rightarrow term) \\ \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \overline{spine\_elem_i}^{\,i} :: arg \gg ret\end{array}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \overline{spine\_elem_i}^{\,i} :: term \supset arg \gg ret}\ \textsc{Spine\_Constraint}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi \vdash pexpr \Rightarrow ident{:}\beta.\ term}$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash pval \Rightarrow y{:}\beta.\ term}{\mathcal{C};\mathcal{L};\Phi \vdash pval \Rightarrow y{:}\beta.\ term}\ \textsc{PExpr\_Val}$$

$$\frac{\begin{array}{c}\mathcal{C};\mathcal{L};\Phi \vdash pval_1 \Rightarrow \_'{:}\texttt{loc}.\ \_' \\ \mathcal{C};\mathcal{L};\Phi \vdash pval_2 \Rightarrow \_''{:}\texttt{integer}.\ \_''\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{array\_shift}\,(pval_1, \tau, pval_2) \Rightarrow y{:}\texttt{loc}.\ y = pval_1 +_{\text{ptr}} (pval_2 \times \text{size\_of}(\tau))}\ \textsc{PExpr\_Array\_Shift}$$

$$\frac{\begin{array}{c}\mathcal{C};\mathcal{L};\Phi \vdash pval \Rightarrow \_{:}\texttt{loc}.\ \_ \\ \_'{:}\,\texttt{struct}\,tag\ \&\,\overline{member_i{:}\tau_i}^{\,i} \in \texttt{Globals}\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{member\_shift}\,(pval, tag, member_j) \Rightarrow y{:}\texttt{loc}.\ y = pval +_{\text{ptr}} \text{offset\_of}_{tag}(member_j)}\ \textsc{PExpr\_Member\_Shift}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash pval \Rightarrow \_'{:}\texttt{bool}.\ \_'}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{not}\,(pval) \Rightarrow y{:}\texttt{bool}.\ y = \neg\, pval}\ \textsc{PExpr\_Not}$$

$$\frac{\begin{array}{c}\mathcal{C};\mathcal{L};\Phi \vdash pval_1 \Rightarrow \_1{:}\texttt{integer}.\ \_1 \\ \mathcal{C};\mathcal{L};\Phi \vdash pval_2 \Rightarrow \_2{:}\texttt{integer}.\ \_2\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash pval_1\ binop_{arith}\ pval_2 \Rightarrow y{:}\texttt{integer}.\ y = (pval_1\ binop_{arith}\ pval_2)}\ \textsc{PExpr\_Arith\_Binop}$$

$$\frac{\begin{array}{c}\mathcal{C};\mathcal{L};\Phi \vdash pval_1 \Rightarrow \_1{:}\texttt{integer}.\ \_1 \\ \mathcal{C};\mathcal{L};\Phi \vdash pval_2 \Rightarrow \_2{:}\texttt{integer}.\ \_2\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash pval_1\ binop_{rel}\ pval_2 \Rightarrow y{:}\texttt{bool}.\ y = (pval_1\ binop_{rel}\ pval_2)}\ \textsc{PExpr\_Rel\_Binop}$$

$$\frac{\begin{array}{c}\mathcal{C};\mathcal{L};\Phi \vdash pval_1 \Rightarrow \__1{:}\texttt{bool}.\ \__1 \\ \mathcal{C};\mathcal{L};\Phi \vdash pval_2 \Rightarrow \__2{:}\texttt{bool}.\ \__2\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash pval_1\ binop_{bool}\ pval_2 \Rightarrow y{:}\texttt{bool}.\ y = (pval_1\ binop_{bool}\ pval_2)}\ \textsc{PExpr\_Bool\_Binop}$$

$$\frac{\begin{array}{c}\mathcal{C};\mathcal{L};\Phi \vdash pval \Rightarrow \_{:}\texttt{bool}.\ \_ \\ \texttt{smt}\,(\Phi \Rightarrow pval)\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{assert\_undef}\,(pval,\ UB\_name) \Rightarrow y{:}\texttt{unit}.\ y = \texttt{unit}}\ \textsc{PExpr\_Assert\_Undef}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash pval \Rightarrow \_{:}\texttt{bool}.\ \_}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{bool\_to\_integer}\,(pval) \Rightarrow y{:}\texttt{integer}.\ y = \texttt{if}\ pval\ \texttt{then}\ 1\ \texttt{else}\ 0}\ \textsc{PExpr\_Bool\_To\_Integer}$$

$$\frac{\begin{array}{c}\mathcal{C};\mathcal{L};\Phi \vdash pval \Rightarrow \_{:}\texttt{integer}.\ \_ \\ abbrev_1 \equiv \mathrm{max\_int}_\tau - \mathrm{min\_int}_\tau + 1 \\ abbrev_2 \equiv pval\ \texttt{rem\_f}\ abbrev_1\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{wrapI}\,(\tau, pval) \Rightarrow y'{:}\beta.\ y = \texttt{if}\ abbrev_2 \leq \mathrm{max\_int}_\tau\ \texttt{then}\ abbrev_2\ \texttt{else}\ abbrev_2 - abbrev_1}\ \textsc{PExpr\_WrapI}$$

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

$$\frac{}{\_{:}\beta{:}\beta \rightsquigarrow \cdot}\ \textsc{Pattern\_No\_Sym\_Annot}$$

$$\frac{}{x{:}\beta{:}\beta \rightsquigarrow \cdot, x{:}\beta}\ \textsc{Pattern\_Sym\_Annot}$$

$$\frac{}{\texttt{Nil}\ \beta(\,){:}[\beta] \rightsquigarrow \cdot}\ \textsc{Pattern\_Nil}$$

$$\frac{\begin{array}{c}pattern_1{:}\beta \rightsquigarrow \mathcal{C}_1 \\ pattern_2{:}[\beta] \rightsquigarrow \mathcal{C}_2\end{array}}{\texttt{Cons}(pattern_1, pattern_2){:}[\beta] \rightsquigarrow \mathcal{C}_1, \mathcal{C}_2}\ \textsc{Pattern\_Cons}$$

$$\frac{\overline{pattern_i{:}\beta_i \rightsquigarrow \mathcal{C}_i}^{\,i}}{\texttt{Tuple}(\overline{pattern_i}^{\,i}){:}\overline{\beta_i}^{\,i} \rightsquigarrow \overline{\mathcal{C}_i}^{\,i}} \quad \textsc{Pattern\_Tuple}$$

$$\frac{\overline{pattern_i{:}\beta \rightsquigarrow \mathcal{C}_i}^{\,i}}{\texttt{Array}(\overline{pattern_i}^{\,i}){:}\texttt{array}\,\beta \rightsquigarrow \overline{\mathcal{C}_i}^{\,i}} \quad \textsc{Pattern\_Array}$$

$$\frac{pattern{:}\beta \rightsquigarrow \mathcal{C}}{\texttt{Specified}(pattern){:}\beta \rightsquigarrow \mathcal{C}} \quad \textsc{Pattern\_Specified}$$

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

$$\frac{}{x{:}\beta \rightsquigarrow \cdot, x{:}\beta} \quad \textsc{Sym\_Or\_PatternSym}$$

$$\frac{pattern{:}\beta \rightsquigarrow \mathcal{C}}{pattern{:}\beta \rightsquigarrow \mathcal{C}} \quad \textsc{Sym\_Or\_PatternPattern}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi \vdash tpval \Leftarrow ident{:}\beta.\, term}$

$$\frac{\texttt{smt}\,(\Phi \Rightarrow \texttt{false})}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{undef}\ \ UB\_name \Leftarrow y{:}\beta.\, term} \quad \textsc{TPVal\_Undef}$$

$$\frac{\begin{array}{l}\mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow y{:}\beta.\, term' \\ \texttt{smt}\,(\Phi, term' \Rightarrow term)\end{array}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{done}\, pval \Leftarrow y{:}\beta.\, term} \quad \textsc{TPVal\_Done}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi \vdash tpexpr \Leftarrow ident{:}\beta.\, term}$

$$\frac{\begin{array}{l}\mathcal{C};\mathcal{L};\Phi \vdash pval \Rightarrow \_{:}\texttt{bool}.\ \_ \\ \mathcal{C};\mathcal{L};\Phi, pval = \texttt{true} \vdash tpexpr_1 \Leftarrow y{:}\beta.\ term \\ \mathcal{C};\mathcal{L};\Phi, pval = \texttt{false} \vdash tpexpr_2 \Leftarrow y{:}\beta.\ term\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{if}\ pval\ \texttt{then}\ tpexpr_1\ \texttt{else}\ tpexpr_2 \Leftarrow y{:}\beta.\ term}\ \textsc{TPExpr\_If}$$

$$\frac{\begin{array}{l}\mathcal{C};\mathcal{L};\Phi \vdash pexpr \Rightarrow \_{:}\beta.\ \_ \\ ident\_or\_pattern{:}\beta \rightsquigarrow \mathcal{C}' \\ \mathcal{C}, \mathrm{fresh}(\mathcal{C}');\mathcal{L};\Phi \vdash [\mathrm{fresh}(\mathcal{C}')/\mathcal{C}']tpexpr \Leftarrow y{:}\beta.\ term\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{let}\ ident\_or\_pattern = pexpr\ \texttt{in}\ tpexpr \Leftarrow y{:}\beta.\ term}\ \textsc{TPExpr\_Let}$$

$$\frac{\begin{array}{l}\mathcal{C};\mathcal{L};\Phi \vdash pval \Rightarrow \_{:}\beta.\ \_ \\ \overline{pattern_i{:}\beta \rightsquigarrow \mathcal{C}_i}^{\,i} \\ \overline{\mathcal{C}, \mathrm{fresh}(\mathcal{C}_i);\mathcal{L};\Phi \vdash [\mathrm{fresh}(\mathcal{C}_i)/\mathcal{C}_i]tpexpr_i \Leftarrow y{:}\beta.\ term}^{\,i}\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{case}\ pval\ \texttt{of}\ \overline{\mid\ pattern_i \Rightarrow tpexpr_i}^{\,i}\ \texttt{end} \Leftarrow y{:}\beta.\ term}\ \textsc{TPExpr\_Case}$$

$\boxed{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash mem\_action \Rightarrow ret}$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash pval \Rightarrow \_{:}\texttt{integer}.\ \_}{\mathcal{C};\mathcal{L};\Phi;\cdot \vdash \texttt{create}\,(pval, \tau) \Rightarrow \Sigma\, y_p{:}\texttt{loc}.\ \exists\, y{:}\beta_\tau.\ \texttt{representable}\,(\tau*, y_p) \wedge \texttt{alignedI}\,(pval, y_p) \wedge [y_p]\ 1 \overset{\times}{\mapsto}_\tau [y] \star \mathtt{I}}\ \textsc{Action\_Create}$$

$$\frac{\begin{array}{l}\mathcal{C};\mathcal{L};\Phi \vdash pval_1 \Rightarrow \_{:}\texttt{loc}.\ \_ \\ \mathcal{C};\mathcal{L};\Phi \vdash pval' \Rightarrow y'{:}\beta_\tau.\ term' \\ \texttt{smt}\,(\Phi, term' \Rightarrow \texttt{representable}\,(\tau, y')) \\ \texttt{smt}\,(\Phi \Rightarrow pval_0 = pval_1)\end{array}}{\mathcal{C};\mathcal{L};\Phi;\cdot, [pval_0]\ 1 \mapsto_\tau [\_''] \vdash \texttt{store}\,(\_, \tau, pval_1, pval', \_) \Rightarrow \Sigma\, \_{:}\texttt{unit}.\ \exists\, y'{:}\beta_\tau.\ term' \wedge [pval_0]\ 1 \overset{\checkmark}{\mapsto}_\tau [y'] \star \mathtt{I}}\ \textsc{Action\_Store}$$

$$\frac{\begin{array}{l}\mathcal{C};\mathcal{L};\Phi \vdash pval_1 \Rightarrow \_{:}\texttt{loc}.\ \_ \\ \texttt{smt}\,(\Phi \Rightarrow pval_0 = pval_1)\end{array}}{\mathcal{C};\mathcal{L};\Phi;\cdot, [pval_0]\ 1 \mapsto_\tau [\_'] \vdash \texttt{kill}\,(\texttt{static}\,\tau, pval_1) \Rightarrow \Sigma\, \_{:}\texttt{unit}.\ \mathtt{I}}\ \textsc{Action\_Kill\_Static}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{seq\_expr} \Rightarrow \mathit{ret}}$

$$\frac{\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pval} \Rightarrow y{:}\beta.\ \mathit{term}}{\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash \mathit{pval} \Rightarrow \Sigma\, y{:}\beta.\ \mathit{term} \wedge \mathtt{I}} \quad \textsc{Seq\_Expr\_Pure}$$

$\boxed{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{is\_expr} \Rightarrow \mathit{ret}}$

$$\frac{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{mem\_action} \Rightarrow \mathit{ret}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathtt{Pos}\, \mathit{mem\_action} \Rightarrow \mathit{ret}} \quad \textsc{IS\_Expr\_Action}$$

$$\frac{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{mem\_action} \Rightarrow \mathit{ret}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathtt{Neg}\, \mathit{mem\_action} \Rightarrow \mathit{ret}} \quad \textsc{IS\_Expr\_Neg\_Action}$$

```
Definition rules:         54 good    0 bad
Definition rule clauses: 123 good    0 bad
```