ident, x, y, y_p, _	subscript p is for pointers
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
n, i	index variables
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
	|	τ*		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
	|	$\beta_1 \times .. \times \beta_n$		tuple
	|	`struct` *tag*		struct
	|	$\{\beta\}$		set
	|	`opt` (β)		option
	|	$\beta_1, .., \beta_n \rightarrow \beta$		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

	| `>`	greater than
	| `<`	less than
	| `>=`	greater than or equal to
	| `<=`	less than or equal to
	| `/\`	conjucntion
	| `\/`	disjunction
object_value	::=	C object values (inhabitants of object types), which can be read/stored
	| *mem_int*	integer value
	| *mem_ptr*	pointer value
	| `array` $(\overline{loaded\_value_i}^{\,i})$	C array value
	| $(\texttt{struct}\, ident)\{\,\overline{.member_i{:}\tau_i = mem\_val_i}^{\,i}\,\}$	C struct value
	| $(\texttt{union}\, ident)\{\,.member = mem\_val\}$	C union value
smt_object_value	::=	like above, but can be embeded into the SMT value grammar
	| *mem_int*	integer value
	| *mem_ptr*	pointer value
	| $(\texttt{struct}\, ident)\{\,\overline{.member_i{:}\tau_i = mem\_val_i}^{\,i}\,\}$	C struct value
	| $(\texttt{union}\, ident)\{\,.member = mem\_val\}$	C union value
loaded_value	::=	potentially unspecified C object values
	| `specified` *object_value*	specified loaded value
smt_loaded_value	::=	like above, but can be embeded into SMT value grammar
	| `specified` *smt_object_value*	specified loaded value
value	::=	Core values
	| *object_value*	C object value
	| *loaded_value*	loaded C object value
	| `Unit`	unit

	`True`	boolean true
	`False`	boolean false
	$\beta[\overline{value_i}^{\,i}]$	list
	$(\overline{value_i}^{\,i})$	tuple

smt_value	::=		like above, but can be embeded into SMT value grammar
	|	*smt_object_value*	C object value
	|	*smt_loaded_value*	loaded C object value
	|	`Unit`	unit
	|	`True`	boolean true
	|	`False`	boolean false
	|	$\beta[\overline{smt\_value_i}^{\,i}]$	list
	|	$(\overline{smt\_value_i}^{\,i})$	tuple

ctor_val	::=		data constructors
	|	`Nil` β	empty list
	|	`Cons`	list cons
	|	`Tuple`	tuple
	|	`Array`	C array
	|	`Specified`	non-unspecified loaded value

smt_ctor_val	::=		like above, but can be embeded into SMT value grammar
	|	`Nil` β	empty list
	|	`Cons`	list cons
	|	`Tuple`	tuple
	|	`Specified`	non-unspecified loaded value

ctor_expr	::=		data constructors
	|	`Ivmax`	max integer value
	|	`Ivmin`	min integer value

	`Ivsizeof`	sizeof value
	`Ivalignof`	alignof value
	`IvCOMPL`	bitwise complement
	`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

smt_pval	::=	like above, but can be embeded into SMT value grammar
	ident	Core identifier
	impl_const	implementation-defined constant
	smt_value	Core values
	`constrained` $(\overline{mem\_iv\_c_i, smt\_pval_i}^{\,i})$	constrained value
	`error` $(string, smt\_pval)$	impl-defined static error
	$smt\_ctor\_val(\overline{smt\_pval_i}^{\,i})$	data constructor application

	$\mid$	$(\texttt{struct}\, ident)\{\, \overline{.member_i = smt\_pval_i}^{\,i} \}$	C struct expression
	$\mid$	$(\texttt{union}\, ident)\{\, .member = smt\_pval\}$	C union expression
$pexpr$	::=		pure expressions
	$\mid$	$pval$	pure values
	$\mid$	$ctor\_expr(\overline{pval_i}^{\,i})$	data constructor application
	$\mid$	$\texttt{array\_shift}\,(pval_1, \tau, pval_2)$	pointer array shift
	$\mid$	$\texttt{member\_shift}\,(pval, ident, member)$	pointer struct/union member shift
	$\mid$	$\texttt{not}\,(pval)$	boolean not
	$\mid$	$pval_1\ binop\ pval_2$	binary operations
	$\mid$	$\texttt{memberof}\,(ident, member, pval)$	C struct/union member access
	$\mid$	$name(pval_1, \ldots, pval_n)$	pure function call
	$\mid$	$\texttt{assert\_undef}\,(pval,\ UB\_name)$	
	$\mid$	$\texttt{bool\_to\_integer}\,(pval)$	
	$\mid$	$\texttt{conv\_int}\,(\tau, pval)$	
	$\mid$	$\texttt{wrapI}\,(\tau, pval)$	
$tpval$	::=		top-level pure values
	$\mid$	$\texttt{undef}\ UB\_name$	undefined behaviour
	$\mid$	$\texttt{done}\, pval$	pure done
$ident\_opt\_\beta$	::=		type annotated optional identifier
	$\mid$	$\_{:}\beta$	
	$\mid$	$ident{:}\beta$	
$pattern$	::=		
	$\mid$	$ident\_opt\_\beta$	
	$\mid$	$ctor\_val(\overline{pattern_i}^{\,i})$	
$ident\_or\_pattern$	::=		

	$\mid$	*ident*	
	$\mid$	*pattern*	
tpexpr	::=		top-level pure expressions
	$\mid$	*tpval*	top-level pure values
	$\mid$	$\texttt{case}\, pval\, \texttt{of}\, \overline{\mid pattern_i \Rightarrow tpexpr_i}^{\,i}\, \texttt{end}$	pattern matching
	$\mid$	$\texttt{let}\, ident\_or\_pattern = pexpr\, \texttt{in}\, tpexpr$	pure let
	$\mid$	$\texttt{if}\, pval\, \texttt{then}\, tpexpr_1\, \texttt{else}\, tpexpr_2$	pure if
	$\mid$	$[\mathcal{C}/\mathcal{C}']tpexpr$	M simul-sub all vars in $\mathcal{C}$ for all vars in $\mathcal{C}'$ in *tpexpr*
m_kill_kind	::=		
	$\mid$	`dynamic`	
	$\mid$	$\texttt{static}\, \tau$	
bool, _	::=		OCaml booleans
	$\mid$	`true`	
	$\mid$	`false`	
int, _	::=		OCaml fixed-width integer
	$\mid$	i	literal integer
mem_action	::=		memory actions
	$\mid$	$\texttt{create}\, (pval, \tau)$	
	$\mid$	$\texttt{create\_readonly}\, (pval_1, \tau, pval_2)$	
	$\mid$	$\texttt{alloc}\, (pval_1, pval_2)$	
	$\mid$	$\texttt{kill}\, (m\_kill\_kind, pval)$	
	$\mid$	$\texttt{store}\, (bool, \tau, pval_1, pval_2, mem\_order)$	true means store is locking
	$\mid$	$\texttt{load}\, (\tau, pval, mem\_order)$	
	$\mid$	$\texttt{rmw}\, (\tau, pval_1, pval_2, pval_3, mem\_order_1, mem\_order_2)$	
	$\mid$	$\texttt{fence}\, (mem\_order)$	

		$\texttt{cmp\_exch\_strong}\,(\tau, pval_1, pval_2, pval_3, mem\_order_1, mem\_order_2)$
		$\texttt{cmp\_exch\_weak}\,(\tau, pval_1, pval_2, pval_3, mem\_order_1, mem\_order_2)$
		$\texttt{linux\_fence}\,(linux\_mem\_order)$
		$\texttt{linux\_load}\,(\tau, pval, linux\_mem\_order)$
		$\texttt{linux\_store}\,(\tau, pval_1, pval_2, linux\_mem\_order)$
		$\texttt{linux\_rmw}\,(\tau, pval_1, pval_2, linux\_mem\_order)$

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
	|	$pval_1 == pval_2$	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
	|	$\texttt{intFromPtr}\,(\tau_1, \tau_2, pval)$	cast of pointer value to integer value
	|	$\texttt{ptrFromInt}\,(\tau_1, \tau_2, pval)$	cast of integer value to pointer value
	|	$\texttt{ptrValidForDeref}\,(\tau, pval)$	dereferencing validity predicate
	|	$\texttt{ptrWellAligned}\,(\tau, pval)$	
	|	$\texttt{ptrArrayShift}\,(pval_1, \tau, pval_2)$	
	|	$\texttt{memcpy}\,(pval_1, pval_2, pval_3)$	
	|	$\texttt{memcmp}\,(pval_1, pval_2, pval_3)$	
	|	$\texttt{realloc}\,(pval_1, pval_2, pval_3)$	

	$\mid$	$\texttt{va\_start}\,(pval_1, pval_2)$	
	$\mid$	$\texttt{va\_copy}\,(pval)$	
	$\mid$	$\texttt{va\_arg}\,(pval, \tau)$	
	$\mid$	$\texttt{va\_end}\,(pval)$	
$tval$	::=		(effectful) top-level values
	$\mid$	$\texttt{done}\,pval$	end of top-level expression
	$\mid$	$\texttt{undef}\ UB\_name$	undefined behaviour
seq_expr	::=		sequential (effectful) expressions
	$\mid$	$pval$	pure values
	$\mid$	$\texttt{ccall}\,(\tau, pval, \overline{pval_i}^{\,i})$	C function call
	$\mid$	$\texttt{pcall}\,(name, \overline{pval_i}^{\,i})$	procedure call
seq_texpr	::=		sequential top-level (effectful) expressions
	$\mid$	$tval$	(effectful) top-level values
	$\mid$	$\texttt{run}\,ident\,pval_1, \ldots, pval_n$	run from label
	$\mid$	$\texttt{nd}\,(pval_1, \ldots, pval_n)$	nondeterministic choice
	$\mid$	$\texttt{let}\,ident\_or\_pattern = seq\_expr\,\texttt{in}\,texpr$	pure sequencing
	$\mid$	$\texttt{case}\,pval\,\texttt{with}\,\overline{\mid pattern_i \Rightarrow texpr_i}^{\,i}\,\texttt{end}$	pattern matching
	$\mid$	$\texttt{if}\,pval\,\texttt{then}\,texpr_1\,\texttt{else}\,texpr_2$	conditional
	$\mid$	$\texttt{bound}\,[int](is\_texpr)$	limit scope of indet seq behaviour, absent at runtime
is_expr	::=		indet seq (effectful) expressions
	$\mid$	$\texttt{memop}\,(mem\_op)$	pointer op involving memory
	$\mid$	pol_mem_action	memory action
	$\mid$	$\texttt{unseq}\,(texpr_1, \ldots, texpr_n)$	unsequenced expressions
is_texpr	::=		indet seq top-level (effectful) expressions
	$\mid$	$\texttt{let weak}\,pattern = is\_expr\,\texttt{in mu\_texpr\_aux}$	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	
	|	$\exists$	
	|	$\star$	
	|	$\times$	
	|	$\wedge$	
	|	$\bigwedge$	
	|	$\neg$	
	|	$=$	
	|	$\neq$	
	|	$\leq$	
	|	$\geq$	

	&
	.
	|
	$+_{\text{ptr}}$
	$\mapsto$
	$*$
	::
	✓
	:
	.
	.
	$\gg$
	::

z	::=			OCaml arbitrary-width integer
	|	i	M	literal integer
	|	of_mem_int(mem_int)	M	
	|	of_ctype(τ)	M	size of a C type
	|	`ptr_size`	M	size of a pointer

$\mathbb{Q}$	::=	OCaml type for rational numbers
	|	$\frac{int_1}{int_2}$

lit	::=
	| *ident*
	| `unit`
	| *bool*
	| z
	| $\mathbb{Q}$

$$
\begin{array}{lll}
\mathit{bool\_op} & ::= & \\
 & | & \neg\, \mathit{term} \\
 & | & \mathit{term}_1 = \mathit{term}_2 \\
 & | & \bigwedge(\overline{\mathit{term}_i}^{\,i}) \\
 & & \\
\mathit{arith\_op} & ::= & \\
 & | & \mathit{term}_1 \times \mathit{term}_2 \\
 & & \\
\mathit{list\_op} & ::= & \\
 & | & \texttt{nil} \\
 & | & \texttt{tl}\, \mathit{term} \\
 & | & \mathit{term}^{(\mathit{int})} \\
 & & \\
\mathit{tuple\_op} & ::= & \\
 & | & (\overline{\mathit{term}_i}^{\,i}) \\
 & | & \mathit{term}^{(\mathit{int})} \\
 & & \\
\mathit{pointer\_op} & ::= & \\
 & | & \texttt{of\_mem\_ptr}\, \mathit{mem\_ptr} \\
 & | & \mathit{term}_1 +_{\mathrm{ptr}} \mathit{term}_2 \\
 & & \\
\mathit{option\_op} & ::= & \\
 & | & \texttt{none}\, \mathit{BT\_t} \\
 & | & \texttt{some}\, \mathit{term} \\
 & & \\
\mathit{array\_op} & ::= & \\
 & | & \mathit{term}_1[\mathit{term}_2] \\
 & & \\
\mathit{param\_op} & ::= & \\
 & | & \mathit{term}(\mathit{term}_1, .., \mathit{term}_n)
\end{array}
$$

struct_op	::=			
	|	$term.member$		
ct_pred	::=			
	|	`representable` $(\tau, term)$		
	|	`alignedI` $(term_1, term_2)$		
term, _	::=			
	|	*lit*		
	|	*arith_op*		
	|	*bool_op*		
	|	*tuple_op*		
	|	*struct_op*		
	|	*pointer_op*		
	|	*list_op*		
	|	*array_op*		
	|	*ct_pred*		
	|	*option_op*		
	|	*param_op*		
	|	$(term)$	S	parentheses
	|	$[term_1/ident]term_2$	M	substitute $term_1$ for *ident* in $term_2$
	|	*smt_pval*	M	can be embeded into the SMT value grammar
	|	*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
	$\mid$	$\checkmark$	initialised
	$\mid$	$\times$	uninitalised
$predicate$	$::=$		arbitrary predicate
	$\mid$	$terms_1 \, \mathbb{Q}^{init}_{\mapsto \, predicate\_name} \, terms_2$	
$resource$	$::=$		
	$\mid$	$predicate$	
$spine\_elem$	$::=$		spine element
	$\mid$	$pval$	pure value
	$\mid$	$logical\_val$	logical variable
	$\mid$	$resource$	resource
arg	$::=$		argument types
	$\mid$	$\Pi \, ident{:}\beta.\ arg$	
	$\mid$	$\forall \, ident{:}\beta.\ arg$	
	$\mid$	$resource \multimap arg$	
	$\mid$	$term \supset arg$	
	$\mid$	ret	
	$\mid$	$[spine\_elem/ident]arg$	M
$ret, \ \_$	$::=$		return types
	$\mid$	$\Sigma \, ident{:}\beta.\ ret$	
	$\mid$	$\exists \, ident{:}\beta.\ ret$	
	$\mid$	$resource \star ret$	
	$\mid$	$term \wedge ret$	
	$\mid$	$\mathtt{I}$	

$\mathcal{C}$	::=			computational var env
	|	.		
	|	$\mathcal{C}, ident{:}BT\_t$		
	|	$\mathcal{C}, \mathcal{C}'$		
	|	$\text{fresh}(\mathcal{C})$	M	identical context except with fresh variable names
$\mathcal{L}$	::=			logical var env
	|	.		
	|	$\mathcal{L}, ident$		
Φ	::=			constraints env
	|	.		
	|	$\Phi, term$		
$\mathcal{R}$	::=			resources env
	|	.		
	|	$\mathcal{R}, resource$		
formula	::=			
	|	*judgement*		
	|	$\texttt{smt}\,(\Phi \Rightarrow term)$		
	|	$ident{:}\beta \in \mathcal{C}$		
	|	$ident{:}\texttt{struct}\,tag\;\&\,\overline{member_i{:}\tau_i}^{\,i} \in \texttt{Globals}$		
	|	$\overline{\mathcal{C}_i; \mathcal{L}_i; \Phi_i \vdash mem\_val_i \Rightarrow \texttt{mem}\,y_i{:}\beta_i.\,term_i}^{\,i}$		dependent on memory object model
	|	$\overline{\mathcal{C}_i; \mathcal{L}_i; \Phi_i \vdash pval_i \Rightarrow ident_i{:}\beta_i.\,term_i}^{\,i}$		
	|	$\overline{pattern_i{:}\beta_i \rightsquigarrow \mathcal{C}_i}^{\,i}$		
	|	$\overline{\mathcal{C}_i; \mathcal{L}_i; \Phi_i \vdash tpexpr_i \Leftarrow y_i{:}\beta_i.\,term_i}^{\,i}$		
	|	$\mathcal{L} \vdash logical\_val{:}\beta$		
object_value_jtype	::=			

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

$pval\_jtype$ ::=

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

$spine\_jtype$ ::=

$$\mid \quad \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash spine\_elem_1, \ldots, spine\_elem_n :: arg \gg ret$$

$pexpr\_jtype$ ::=

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

$pattern\_jtype$ ::=

$$\mid \quad pattern{:}\beta \rightsquigarrow \mathcal{C}$$
$$\mid \quad ident\_or\_pattern{:}\beta \rightsquigarrow \mathcal{C}$$

$tpval\_jtype$ ::=

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

$tpexpr\_jtype$ ::=

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

$action\_jtype$ ::=

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

$judgement$ ::=

$\mid$ $object\_value\_jtype$
$\mid$ $pval\_jtype$
$\mid$ $spine\_jtype$
$\mid$ $pexpr\_jtype$
$\mid$ $pattern\_jtype$

| *tpval_jtype*
| *tpexpr_jtype*
| *action_jtype*

user_syntax ::=
| *ident*
| *impl_const*
| *mem_int*
| *member*
|
| *nat*
| *n*
| *mem_ptr*
| *mem_val*
|
| *mem_iv_c*
| *UB_name*
| *string*
|
|
| *mem_order*
| *linux_mem_order*
|
| *logical_val*
| *Sctypes_t*
| *tag*
| β
| *binop*
| *ident*
| τ

| *ident*
| *object_value*
| *smt_object_value*
| *loaded_value*
| *smt_loaded_value*
| β
| *value*
| *smt_value*
| *ctor_val*
| *smt_ctor_val*
| *ctor_expr*
| τ
| *name*
| *pval*
| *smt_pval*
| *pval*
| *smt_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
| $bool\_op$
| $arith\_op$
| $list\_op$
| $tuple\_op$
| $pointer\_op$
| 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*

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$\boxed{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash spine\_elem_1,\ ..,\ spine\_elem_n :: arg \gg ret}$

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

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{pval} \Rightarrow \_{:}\beta.\ \_ \quad \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{spine\_elem}_1,\ ..,\ \mathit{spine\_elem}_n :: [\mathit{pval}/x]\mathit{arg} \gg \mathit{ret}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{pval}, \mathit{spine\_elem}_1,\ ..,\ \mathit{spine\_elem}_n :: \Pi\, x{:}\beta.\ \mathit{arg} \gg \mathit{ret}}\ \textsc{Spine\_Computational}$$

$$\frac{\mathcal{L} \vdash \mathit{logical\_val}{:}\beta \quad \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{spine\_elem}_1,\ ..,\ \mathit{spine\_elem}_n :: [\mathit{logical\_val}/x]\mathit{arg} \gg \mathit{ret}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{logical\_val}, \mathit{spine\_elem}_1,\ ..,\ \mathit{spine\_elem}_n :: \forall\, x{:}\beta.\ \mathit{arg} \gg \mathit{ret}}\ \textsc{Spine\_Logical}$$

$$\frac{\mathtt{smt}\,(\Phi \Rightarrow \mathit{resource} = \mathit{resource}') \quad \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{spine\_elem}_1,\ ..,\ \mathit{spine\_elem}_n :: \mathit{arg} \gg \mathit{ret}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R}, \mathit{resource} \vdash \mathit{resource}', \mathit{spine\_elem}_1,\ ..,\ \mathit{spine\_elem}_n :: \mathit{resource}' \multimap \mathit{arg} \gg \mathit{ret}}\ \textsc{Spine\_Resource}$$

$$\frac{\mathtt{smt}\,(\Phi \Rightarrow \mathit{term}) \quad \mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{spine\_elem}_1,\ ..,\ \mathit{spine\_elem}_n :: \mathit{arg} \gg \mathit{ret}}{\mathcal{C};\mathcal{L};\Phi;\mathcal{R} \vdash \mathit{spine\_elem}_1,\ ..,\ \mathit{spine\_elem}_n :: \mathit{term} \supset \mathit{arg} \gg \mathit{ret}}\ \textsc{Spine\_Constraint}$$

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

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

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{smt\_pval}_1 \Rightarrow \_'{:}\mathtt{loc}.\ \_' \quad \mathcal{C};\mathcal{L};\Phi \vdash \mathit{smt\_pval}_2 \Rightarrow \_''{:}\mathtt{integer}.\ \_''}{\mathcal{C};\mathcal{L};\Phi \vdash \mathtt{array\_shift}\,(\mathit{smt\_pval}_1, \tau, \mathit{smt\_pval}_2) \Rightarrow y{:}\mathtt{loc}.\ y = \mathit{smt\_pval}_1 +_{\mathrm{ptr}} \mathit{smt\_pval}_2 \times \mathrm{of\_ctype}(\tau)}\ \textsc{PExpr\_Array\_Shift}$$

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

$$\frac{\begin{array}{l}\mathcal{C};\mathcal{L};\Phi \vdash name \Rightarrow \_{:}\beta.\ \_ \\ \mathcal{C};\mathcal{L};\Phi;\cdot \vdash pval_1,\ ..,pval_n :: \forall\ \_{:}\beta.\ \mathtt{I} \gg \Sigma\ y{:}\beta'.\ term' \wedge \mathtt{I}\end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash name(pval_1,\ ..,pval_n) \Rightarrow y{:}\beta'.\ term'}\ \textsc{PExpr\_Call}$$

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\begin{array}{l}\mathcal{C};\mathcal{L};\Phi \vdash smt\_pval_1 \Rightarrow \_{:}\mathtt{loc}.\ \_ \\ \mathcal{C};\mathcal{L};\Phi \vdash smt\_pval' \Rightarrow \_'{:}\mathrm{of\_ctype}(\tau).\ \_' \\ \mathtt{smt}\,(\Phi \Rightarrow \mathtt{representable}\,(\tau, smt\_pval')) \\ \mathtt{smt}\,(\Phi \Rightarrow smt\_pval_0 = smt\_pval_1)\end{array}}{\mathcal{C};\mathcal{L};\Phi;\cdot, [smt\_pval_0]\ 1{\mapsto_\tau}\ [\_''] \vdash \mathtt{store}\,(\_, \tau, smt\_pval_1, smt\_pval', \_) \Rightarrow \Sigma\ \_{:}\mathtt{unit}.\ [smt\_pval_0]\ 1{\overset{\checkmark}{\mapsto}_\tau}\ [smt\_pval'] \star \mathtt{I}}\ \textsc{Action\_Store}$$

$$\frac{\begin{array}{l}\mathcal{C};\mathcal{L};\Phi \vdash smt\_pval_1 \Rightarrow \_{:}\mathtt{loc}.\ \_ \\ \mathtt{smt}\,(\Phi \Rightarrow smt\_pval_0 = smt\_pval_1)\end{array}}{\mathcal{C};\mathcal{L};\Phi;\cdot, [smt\_pval_0]\ 1{\mapsto_\tau}\ [\_'] \vdash \mathtt{kill}\,(\mathtt{static}\,\tau, smt\_pval_1) \Rightarrow \Sigma\ \_{:}\mathtt{unit}.\ \mathtt{I}}\ \textsc{Action\_Kill\_Static}$$

```
Definition rules:         36 good    0 bad
Definition rule clauses:  84 good    0 bad
```