

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

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

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

		<b>True</b>	boolean true
		<b>False</b>	boolean false
		$\beta[\overline{value_i}^i]$	list
		$(\overline{value_i}^i)$	tuple
<i>smt_value</i>	::=		like above, but can be embeded into SMT value grammar
		<i>smt_object_value</i>	C object value
		<i>smt_loaded_value</i>	loaded C object value
		<b>Unit</b>	unit
		<b>True</b>	boolean true
		<b>False</b>	boolean false
		$\beta[\overline{smt\_value_i}^i]$	list
		$(\overline{smt\_value_i}^i)$	tuple
<i>ctor_val</i>	::=		data constructors
		<b>Nil</b> $\beta$	empty list
		<b>Cons</b>	list cons
		<b>Tuple</b>	tuple
		<b>Array</b>	C array
		<b>Specified</b>	non-unspecified loaded value
<i>smt_ctor_val</i>	::=		like above, but can be embeded into SMT value grammar
		<b>Nil</b> $\beta$	empty list
		<b>Cons</b>	list cons
		<b>Tuple</b>	tuple
		<b>Specified</b>	non-unspecified loaded value
<i>ctor_expr</i>	::=		data constructors
		<b>Ivmax</b>	max integer value
		<b>Ivmin</b>	min integer value

		<code>Ivsizeof</code>	sizeof value
		<code>Ivalignof</code>	alignof value
		<code>IvCOMPL</code>	bitwise complement
		<code>IvAND</code>	bitwise AND
		<code>IvOR</code>	bitwise OR
		<code>IvXOR</code>	bitwise XOR
		<code>Fvfromint</code>	cast integer to floating value
		<code>Ivfromfloat</code>	cast floating to integer value
<i>name</i>	::=		
		<i>ident</i>	Core identifier
		<i>impl_const</i>	implementation-defined constant
<i>pval</i>	::=		pure values
		<i>ident</i>	Core identifier
		<i>impl_const</i>	implementation-defined constant
		<i>value</i>	Core values
		<code>constrained</code> ( $\overline{mem\_iv\_c_i, pval_i}^i$ )	constrained value
		<code>error</code> ( <i>string</i> , <i>pval</i> )	impl-defined static error
		<code>ctor_val</code> ( $\overline{pval_i}^i$ )	data constructor application
		<code>(struct</code> <i>ident</i> ) { $\overline{.member_i = pval_i}^i$ }	C struct expression
		<code>(union</code> <i>ident</i> ) { $\overline{.member = pval}$ }	C union expression
<i>smt_pval</i>	::=		like above, but can be embeded into SMT value grammar
		<i>ident</i>	Core identifier
		<i>impl_const</i>	implementation-defined constant
		<i>smt_value</i>	Core values
		<code>constrained</code> ( $\overline{mem\_iv\_c_i, smt\_pval_i}^i$ )	constrained value
		<code>error</code> ( <i>string</i> , <i>smt_pval</i> )	impl-defined static error
		<code>smt_ctor_val</code> ( $\overline{smt\_pval_i}^i$ )	data constructor application

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

	   	<i>ident</i> <i>pattern</i>	
<i>texpr</i>	::=           	<i>tpval</i> <b>case</b> <i>pval</i> <b>of</b> $\overline{\text{pattern}_i \Rightarrow \text{texpr}_i}^i$ <b>end</b> <b>let</b> <i>ident_or_pattern</i> = <i>pexpr</i> <b>in</b> <i>texpr</i> <b>if</b> <i>pval</i> <b>then</b> <i>texpr</i> <sub>1</sub> <b>else</b> <i>texpr</i> <sub>2</sub> [ <i>C/C'</i> ] <i>texpr</i>	top-level pure expressions top-level pure values pattern matching pure let pure if M simul-sub all vars in <i>C</i> for all vars in <i>C'</i> in <i>texpr</i>
<i>m_kill_kind</i>	::=   	<b>dynamic</b> <b>static</b> $\tau$	
<i>bool</i> , $-$	::=   	<b>true</b> <b>false</b>	OCaml booleans
<i>int</i> , $-$	::= 	<i>i</i>	OCaml fixed-width integer literal integer
<i>mem_action</i>	::=               	<b>create</b> ( <i>pval</i> , $\tau$ ) <b>create_readonly</b> ( <i>pval</i> <sub>1</sub> , $\tau$ , <i>pval</i> <sub>2</sub> ) <b>alloc</b> ( <i>pval</i> <sub>1</sub> , <i>pval</i> <sub>2</sub> ) <b>kill</b> ( <i>m_kill_kind</i> , <i>pval</i> ) <b>store</b> ( <i>bool</i> , $\tau$ , <i>pval</i> <sub>1</sub> , <i>pval</i> <sub>2</sub> , <i>mem_order</i> ) <b>load</b> ( $\tau$ , <i>pval</i> , <i>mem_order</i> ) <b>rmw</b> ( $\tau$ , <i>pval</i> <sub>1</sub> , <i>pval</i> <sub>2</sub> , <i>pval</i> <sub>3</sub> , <i>mem_order</i> <sub>1</sub> , <i>mem_order</i> <sub>2</sub> ) <b>fence</b> ( <i>mem_order</i> )	memory actions     true means store is locking

	$\text{cmp\_exch\_strong}(\tau, pval_1, pval_2, pval_3, mem\_order_1, mem\_order_2)$ $\text{cmp\_exch\_weak}(\tau, pval_1, pval_2, pval_3, mem\_order_1, mem\_order_2)$ $\text{linux\_fence}(linux\_mem\_order)$ $\text{linux\_load}(\tau, pval, linux\_mem\_order)$ $\text{linux\_store}(\tau, pval_1, pval_2, linux\_mem\_order)$ $\text{linux\_rmw}(\tau, pval_1, pval_2, linux\_mem\_order)$	
<i>polarity</i>	$::=$ $\text{Pos}$ $\text{Neg}$	polarities for memory actions sequenced by <b>let weak</b> and <b>let strong</b> only sequenced by <b>let strong</b>
<i>pol_mem_action</i>	$::=$ $\text{polarity mem\_action}$	memory actions with polarity
<i>mem_op</i>	$::=$ $pval_1 == pval_2$ $pval_1 \neq pval_2$ $pval_1 < pval_2$ $pval_1 > pval_2$ $pval_1 \leq pval_2$ $pval_1 \geq pval_2$ $pval_1 -_{\tau} pval_2$ $\text{intFromPtr}(\tau_1, \tau_2, pval)$ $\text{ptrFromInt}(\tau_1, \tau_2, pval)$ $\text{ptrValidForDeref}(\tau, pval)$ $\text{ptrWellAligned}(\tau, pval)$ $\text{ptrArrayShift}(pval_1, \tau, pval_2)$ $\text{memcpy}(pval_1, pval_2, pval_3)$ $\text{memcmp}(pval_1, pval_2, pval_3)$ $\text{realloc}(pval_1, pval_2, pval_3)$	operations involving the memory state pointer equality comparison pointer inequality comparison pointer less-than comparison pointer greater-than comparison pointer less-than comparison pointer greater-than comparison pointer subtraction cast of pointer value to integer value cast of integer value to pointer value dereferencing validity predicate



		<b>va_start</b> ( $pval_1, pval_2$ )	
		<b>va_copy</b> ( $pval$ )	
		<b>va_arg</b> ( $pval, \tau$ )	
		<b>va_end</b> ( $pval$ )	
$tval$	::=		(effectful) top-level values
		<b>done</b> $pval$	end of top-level expression
		<b>undef</b> $UB\_name$	undefined behaviour
$seq\_expr$	::=		sequential (effectful) expressions
		$pval$	pure values
		<b>ccall</b> ( $\tau, pval, \overline{pval_i^i}$ )	C function call
		<b>pcall</b> ( $name, \overline{pval_i^i}$ )	procedure call
$seq\_texpr$	::=		sequential top-level (effectful) expressions
		$tval$	(effectful) top-level values
		<b>run</b> $ident\ pval_1, \dots, pval_n$	run from label
		<b>nd</b> ( $pval_1, \dots, pval_n$ )	nondeterministic choice
		<b>let</b> $ident\_or\_pattern = seq\_expr$ <b>in</b> $texpr$	pure sequencing
		<b>case</b> $pval$ <b>with</b> $\overline{pattern_i \Rightarrow texpr_i^i}$ <b>end</b>	pattern matching
		<b>if</b> $pval$ <b>then</b> $texpr_1$ <b>else</b> $texpr_2$	conditional
		<b>bound</b> [ $int$ ] ( $is\_texpr$ )	limit scope of indet seq behaviour, absent at runtime
$is\_expr$	::=		indet seq (effectful) expressions
		<b>memop</b> ( $mem\_op$ )	pointer op involving memory
		$pol\_mem\_action$	memory action
		<b>unseq</b> ( $texpr_1, \dots, texpr_n$ )	unsequenced expressions
$is\_texpr$	::=		indet seq top-level (effectful) expressions
		<b>let weak</b> $pattern = is\_expr$ <b>in</b> $\mu\_texpr\_aux$	weak sequencing

		<code>let strong <i>ident_or_pattern</i> = <i>is_expr</i> in mu_texpr_aux</code>	strong sequencing
<i>texpr</i>	::=		top-level (effectful) expressions
		<i>seq_texpr</i>	sequential (effectful) expressions
		<i>is_texpr</i>	indet seq (effectful) expressions
<i>terminals</i>	::=		
		$\lambda$	
		$\longrightarrow$	
		$\rightarrow$	
		$\rightsquigarrow$	
		$\Rightarrow$	
		$\Leftarrow$	
		$\vdash$	
		$\in$	
		$\Pi$	
		$\forall$	
		$\dashv$	
		$\supset$	
		$\Sigma$	
		$\exists$	
		$\star$	
		$\times$	
		$\wedge$	
		$\bigwedge$	
		$\neg$	
		$=$	
		$\neq$	
		$\leq$	
		$\geq$	

		&	
		.	
		+ <sub>ptr</sub>	
		↦	
		*	
		::	
		✓	
		:	
		.	
		.	
		>>	
		::	
$z$	::=		OCaml arbitrary-width integer
		$i$	M     literal integer
		<code>of_mem_int(<i>mem_int</i>)</code>	M
		<code>of_ctype(<math>\tau</math>)</code>	M     size of a C type
		<code>ptr_size</code>	M     size of a pointer
$\mathbb{Q}$	::=		OCaml type for rational numbers
		$\frac{int_1}{int_2}$	
$lit$	::=		
		<i>ident</i>	
		<code>unit</code>	
		<i>bool</i>	
		$z$	
		$\mathbb{Q}$	

$bool\_op$	$::=$	$\neg term$ $term_1 = term_2$ $\bigwedge(\overline{term_i}^i)$
$arith\_op$	$::=$	$term_1 \times term_2$
$list\_op$	$::=$	$nil$ $tl\ term$ $term^{(int)}$
$tuple\_op$	$::=$	$(\overline{term_i}^i)$ $term^{(int)}$
$pointer\_op$	$::=$	$of\_mem\_ptr\ mem\_ptr$ $term_1 +_{ptr} term_2$
$option\_op$	$::=$	$none\ BT\_t$ $some\ term$
$array\_op$	$::=$	$term_1[term_2]$
$param\_op$	$::=$	$term(term_1, .., term_n)$

<i>struct_op</i>	$::=$   <i>term.member</i>		
<i>ct_pred</i>	$::=$   <b>representable</b> ( $\tau, term$ )   <b>alignedI</b> ( $term_1, term_2$ )		
<i>term, _</i>	$::=$   <i>lit</i>   <i>arith_op</i>   <i>bool_op</i>   <i>tuple_op</i>   <i>struct_op</i>   <i>pointer_op</i>   <i>list_op</i>   <i>array_op</i>   <i>ct_pred</i>   <i>option_op</i>   <i>param_op</i>   ( <i>term</i> )   [ <i>term</i> <sub>1</sub> / <i>ident</i> ] <i>term</i> <sub>2</sub>   <i>smt_pval</i>   <i>resource</i>	S M M	parentheses substitute <i>term</i> <sub>1</sub> for <i>ident</i> in <i>term</i> <sub>2</sub> can be embeded into the SMT value grammar
<i>terms</i>	$::=$   [ <i>term</i> <sub>1</sub> , ..., <i>term</i> <sub><i>n</i></sub> ]		non-empty list of terms
<i>predicate_name</i>	$::=$   <i>Sctypes_t</i>   <i>string</i>		names of predicates C type arbitrary

$init,$	$::=$	initialisation status
	$\checkmark$	initialised
	$\times$	uninitialised
$predicate$	$::=$	arbitrary predicate
	$terms_1 \mathbb{Q} \stackrel{init}{\mapsto} predicate\_name terms_2$	
$resource$	$::=$	
	$predicate$	
$spine\_elem$	$::=$	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
	$\Sigma ident:\beta. ret$	
	$\exists ident:\beta. ret$	
	$resource \star ret$	
	$term \wedge ret$	
	$\mathbf{I}$	

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

		$\mathcal{C}; \mathcal{L}; \Phi \vdash \text{object\_value} \Rightarrow \text{obj ident}:\beta. \text{term}$
$pval\_jtype$	$::=$	$\mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow \text{ident}:\beta. \text{term}$
$spine\_jtype$	$::=$	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{\text{spine\_elem}_i}^i :: \text{arg} \gg \text{ret}$
$pexpr\_jtype$	$::=$	$\mathcal{C}; \mathcal{L}; \Phi \vdash pexpr \Rightarrow \text{ident}:\beta. \text{term}$
$pattern\_jtype$	$::=$	$pattern:\beta \rightsquigarrow \mathcal{C}$   $\text{ident\_or\_pattern}:\beta \rightsquigarrow \mathcal{C}$
$tpval\_jtype$	$::=$	$\mathcal{C}; \mathcal{L}; \Phi \vdash tpval \Leftarrow \text{ident}:\beta. \text{term}$
$tpexpr\_jtype$	$::=$	$\mathcal{C}; \mathcal{L}; \Phi \vdash tpexpr \Leftarrow \text{ident}:\beta. \text{term}$
$action\_jtype$	$::=$	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \text{mem\_action} \Rightarrow \text{ret}$   $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \text{seq\_expr} \Rightarrow \text{ret}$   $\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \text{is\_expr} \Rightarrow \text{ret}$
$judgement$	$::=$	$\text{object\_value\_jtype}$   $pval\_jtype$   $spine\_jtype$



		<i>pexpr_jtype</i>
		<i>pattern_jtype</i>
		<i>tpval_jtype</i>
		<i>tpexpr_jtype</i>
		<i>action_jtype</i>
<i>user_syntax</i>	::=	
		<i>ident</i>
		<i>impl_const</i>
		<i>mem_int</i>
		<i>member</i>
		<i>nat</i>
		<i>n</i>
		<i>mem_ptr</i>
		<i>mem_val</i>
		<i>mem_iv_c</i>
		<i>UB_name</i>
		<i>string</i>
		<i>mem_order</i>
		<i>linux_mem_order</i>
		<i>logical_val</i>
		<i>Sctypes_t</i>
		<i>tag</i>
		$\beta$
		<i>binop</i>

- | *ident*
- |  $\tau$
- | *ident*
- | *object\_value*
- | *smt\_object\_value*
- | *loaded\_value*
- | *smt\_loaded\_value*
- |  $\beta$
- | *value*
- | *smt\_value*
- | *ctor\_val*
- | *smt\_ctor\_val*
- | *ctor\_expr*
- |  $\tau$
- | *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*

$option\_op$   
 $array\_op$   
 $param\_op$   
 $struct\_op$   
 $ct\_pred$   
 $term$   
 $term$   
 $term$   
 $terms$   
 $predicate\_name$   
 $init$   
 $predicate$   
 $resource$   
 $spine\_elem$   
 $arg$   
 $ret$   
 $\mathcal{C}$   
 $\mathcal{L}$   
 $\Phi$   
 $\mathcal{R}$   
 $formula$

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

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash mem\_int \Rightarrow obj\ y:integer. y = of\_mem\_int(mem\_int)} \quad PVAL\_OBJ\_INT$$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash mem\_ptr \Rightarrow obj\ y:loc. y = of\_mem\_ptr\ mem\_ptr} \quad PVAL\_OBJ\_PTR$$

$$\frac{\frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash loaded\_value_i \Rightarrow y_i:\beta. term_i^i}}{\mathcal{C}; \mathcal{L}; \Phi \vdash array(loaded\_value_i^i) \Rightarrow obj\ y:array\ \beta. \bigwedge ([y[i]/y_i] term_i^i)} \quad PVAL\_OBJ\_ARR$$

$$\frac{\frac{\text{ident}:\mathbf{struct\ tag} \ \& \ \overline{\text{member}_i:\tau_i}^i \in \mathbf{Globals}}{\mathcal{C};\mathcal{L};\Phi \vdash \text{mem\_val}_i \Rightarrow \overline{\mathbf{mem\ }y_i:\beta_i.\text{term}_i}^i}}{\mathcal{C};\mathcal{L};\Phi \vdash (\mathbf{struct\ tag})\{\overline{\text{member}_i:\tau_i = \text{mem\_val}_i}^i\} \Rightarrow \mathbf{obj\ }y:\mathbf{struct\ tag}. \bigwedge ([y.\text{member}_i/y_i]\overline{\text{term}_i}^i)} \quad \text{PVAL\_OBJ\_STRUCT}$$

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

$$\frac{x:\beta \in \mathcal{C}}{\mathcal{C};\mathcal{L};\Phi \vdash x \Rightarrow y:\beta. y = x} \quad \text{PVAL\_VAR}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \text{object\_value} \Rightarrow \mathbf{obj\ }y:\beta.\text{term}}{\mathcal{C};\mathcal{L};\Phi \vdash \text{object\_value} \Rightarrow y:\beta.\text{term}} \quad \text{PVAL\_OBJ}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \text{object\_value} \Rightarrow \mathbf{obj\ }y:\beta.\text{term}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathbf{specified\ object\_value} \Rightarrow y:\beta.\text{term}} \quad \text{PVAL\_LOADED}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \mathbf{Unit} \Rightarrow y:\mathbf{unit}. y = \mathbf{unit}} \quad \text{PVAL\_UNIT}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \mathbf{True} \Rightarrow y:\mathbf{bool}. y = \mathbf{true}} \quad \text{PVAL\_TRUE}$$

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \mathbf{False} \Rightarrow y:\mathbf{bool}. y = \mathbf{false}} \quad \text{PVAL\_FALSE}$$

$$\frac{\overline{\mathcal{C};\mathcal{L};\Phi \vdash \text{value}_i \Rightarrow y_i:\beta.\text{term}_i}^i}{\mathcal{C};\mathcal{L};\Phi \vdash \beta[\overline{\text{value}_i}^i] \Rightarrow y:[\beta]. \bigwedge ([y^{(i)}/y_i]\overline{\text{term}_i}^i)} \quad \text{PVAL\_LIST}$$

$$\frac{\overline{\mathcal{C};\mathcal{L};\Phi \vdash \text{value}_i \Rightarrow y_i:\beta_i.\text{term}_i}^i}{\mathcal{C};\mathcal{L};\Phi \vdash (\overline{\text{value}_i}^i) \Rightarrow y:\overline{\beta_i}^i. \bigwedge ([y^{(i)}/y_i]\overline{\text{term}_i}^i)} \quad \text{PVAL\_TUPLE}$$

$$\frac{\text{smt}(\Phi \Rightarrow \text{false})}{\mathcal{C}; \mathcal{L}; \Phi \vdash \text{error}(\text{string}, pval) \Rightarrow y:\beta. \text{term}} \quad \text{PVAL\_ERROR}$$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi \vdash \text{Nil } \beta() \Rightarrow y:[\beta]. y = \text{nil}} \quad \text{PVAL\_CTOR\_NIL}$$

$$\frac{\begin{array}{c} \mathcal{C}; \mathcal{L}; \Phi \vdash pval_1 \Rightarrow y_1:\beta. \text{term}_1 \\ \mathcal{C}; \mathcal{L}; \Phi \vdash pval_2 \Rightarrow y_2:[\beta]. \text{term}_2 \end{array}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \text{Cons}(pval_1, pval_2) \Rightarrow y:[\beta]. [y^{(1)}/y_1] \text{term}_1 \wedge [\text{tl } y/y_2] \text{term}_2} \quad \text{PVAL\_CTOR\_CONS}$$

$$\frac{\overline{\mathcal{C}; \mathcal{L}; \Phi \vdash pval_i \Rightarrow y_i:\beta_i. \text{term}_i^i}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \text{Tuple}(\overline{pval_i^i}) \Rightarrow y:\overline{\beta_i^i}. \bigwedge ([y^{(i)}/y_i] \text{term}_i^i)} \quad \text{PVAL\_CTOR\_TUPLE}$$

$$\frac{\overline{\mathcal{C}; \mathcal{L}; \Phi \vdash pval_i \Rightarrow y_i:\beta. \text{term}_i^i}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \text{Array}(\overline{pval_i^i}) \Rightarrow y:\text{array } \beta. \bigwedge ([y[i]/y_i] \text{term}_i^i)} \quad \text{PVAL\_CTOR\_ARRAY}$$

$$\frac{\mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow y:\beta. \text{term}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \text{Specified}(pval) \Rightarrow y:\beta. \text{term}} \quad \text{PVAL\_CTOR\_SPECIFIED}$$

$$\frac{\overline{\mathcal{C}; \mathcal{L}; \Phi \vdash pval_i \Rightarrow y_i:\beta_i. \text{term}_i^i}}{\mathcal{C}; \mathcal{L}; \Phi \vdash (\text{struct } tag)\{.member_i = pval_i^i\} \Rightarrow y:\text{struct } tag. \bigwedge ([y.member_i/y_i] \text{term}_i^i)} \quad \text{PVAL\_STRUCT}$$

$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{spine\_elem_i^i} :: \text{arg} \gg \text{ret}$

$$\frac{}{\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash :: \text{ret} \gg \text{ret}} \quad \text{SPINE\_EMPTY}$$

$$\frac{\begin{array}{c} \mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow \_:\beta. \_ \\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{spine\_elem_i}^i :: [pval/x]arg \gg ret \end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash pval, \overline{spine\_elem_i}^i :: \Pi x:\beta. arg \gg ret} \text{SPINE\_COMPUTATIONAL}$$

$$\frac{\begin{array}{c} \mathcal{L} \vdash logical\_val:\beta \\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{spine\_elem_i}^i :: [logical\_val/x]arg \gg ret \end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash logical\_val, \overline{spine\_elem_i}^i :: \forall x:\beta. arg \gg ret} \text{SPINE\_LOGICAL}$$

$$\frac{\begin{array}{c} \text{smt}(\Phi \Rightarrow resource = resource') \\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{spine\_elem_i}^i :: arg \gg ret \end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R}, resource \vdash resource', \overline{spine\_elem_i}^i :: resource' \multimap arg \gg ret} \text{SPINE\_RESOURCE}$$

$$\frac{\begin{array}{c} \text{smt}(\Phi \Rightarrow term) \\ \mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{spine\_elem_i}^i :: arg \gg ret \end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \overline{spine\_elem_i}^i :: term \supset arg \gg ret} \text{SPINE\_CONSTRAINT}$$

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

$$\frac{\mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow y:\beta. term}{\mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow y:\beta. term} \text{PEXPR\_VAL}$$

$$\frac{\begin{array}{c} \mathcal{C}; \mathcal{L}; \Phi \vdash smt\_pval_1 \Rightarrow \_':loc. \_ ' \\ \mathcal{C}; \mathcal{L}; \Phi \vdash smt\_pval_2 \Rightarrow \_":integer. \_ '' \end{array}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \text{array\_shift}(smt\_pval_1, \tau, smt\_pval_2) \Rightarrow y:loc. y = smt\_pval_1 +_{\text{ptr}} smt\_pval_2 \times \text{of\_ctype}(\tau)} \text{PEXPR\_ARRAY\_SHIFT}$$

$$\frac{\mathcal{C}; \mathcal{L}; \Phi \vdash smt\_pval \Rightarrow \_':bool. \_ '}{\mathcal{C}; \mathcal{L}; \Phi \vdash \text{not}(smt\_pval) \Rightarrow y:bool. y = \neg smt\_pval} \text{PEXPR\_NOT}$$

$$\begin{array}{c}
\mathcal{C}; \mathcal{L}; \Phi \vdash \text{name} \Rightarrow \_:\beta. \_ \\
\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash pval_1, \dots, pval_n :: \forall \_:\beta. \mathbf{I} \gg \Sigma y:\beta'. \text{term}' \wedge \mathbf{I} \\
\hline
\mathcal{C}; \mathcal{L}; \Phi \vdash \text{name}(pval_1, \dots, pval_n) \Rightarrow y:\beta'. \text{term}'
\end{array}
\quad \text{PEXPR\_CALL}$$

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

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

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

$$\frac{\text{smt}(\Phi \Rightarrow \text{false})}{\mathcal{C}; \mathcal{L}; \Phi \vdash \text{undef } UB\_name \Leftarrow y:\beta. \text{term}}
\quad \text{TPVAL\_UNDEF}$$

$$\frac{\begin{array}{c} \mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow y:\beta. \text{term}' \\ \text{smt}(\Phi, \text{term}' \Rightarrow \text{term}) \end{array}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \text{done } pval \Leftarrow y:\beta. \text{term}}
\quad \text{TPVAL\_DONE}$$

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

$$\frac{\begin{array}{c} \mathcal{C}; \mathcal{L}; \Phi \vdash \text{smt\_pval} \Rightarrow \_:\text{bool}. \_ \\ \mathcal{C}; \mathcal{L}, y'; \Phi, \text{smt\_pval} = \text{true} \vdash \text{tpepr}_1 \Leftarrow y:\beta. \text{term} \\ \mathcal{C}; \mathcal{L}, y'; \Phi, \text{smt\_pval} = \text{false} \vdash \text{tpepr}_2 \Leftarrow y:\beta. \text{term} \end{array}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \text{if } \text{smt\_pval} \text{ then } \text{tpepr}_1 \text{ else } \text{tpepr}_2 \Leftarrow y:\beta. \text{term}}
\quad \text{TPEXPR\_IF}$$

$$\frac{\begin{array}{c} \mathcal{C}; \mathcal{L}; \Phi \vdash \text{pexpr} \Rightarrow \_:\beta. \_ \\ \text{ident\_or\_pattern}:\beta \rightsquigarrow \mathcal{C}' \\ \mathcal{C}, \text{fresh}(\mathcal{C}'); \mathcal{L}; \Phi \vdash [\text{fresh}(\mathcal{C}')/\mathcal{C}'] \text{tpepr} \Leftarrow y:\beta. \text{term} \end{array}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \text{let } \text{ident\_or\_pattern} = \text{pexpr} \text{ in } \text{tpepr} \Leftarrow y:\beta. \text{term}}
\quad \text{TPEXPR\_LET}$$



$$\begin{array}{c}
\mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow \cdot : \beta. \_ \\
\hline
pattern_i : \beta \rightsquigarrow \mathcal{C}_i^i \\
\hline
\mathcal{C}, \text{fresh}(\mathcal{C}_i); \mathcal{L}; \Phi \vdash [\text{fresh}(\mathcal{C}_i)/\mathcal{C}_i] tpepr_i \Leftarrow y : \beta. term^i \\
\hline
\mathcal{C}; \mathcal{L}; \Phi \vdash \text{case } pval \text{ of } \overline{[pattern_i \Rightarrow tpepr_i]^i} \text{ end} \Leftarrow y : \beta. term
\end{array}
\quad \text{TPEXPR\_CASE}$$

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

$$\begin{array}{c}
\mathcal{C}; \mathcal{L}; \Phi \vdash smt\_pval \Rightarrow \cdot : \text{integer}. \_ \\
\hline
\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash \text{create}(smt\_pval, \tau) \Rightarrow \Sigma y_p : \text{loc}. \exists y : \text{of\_ctype}(\tau). \text{representable}(\tau*, y_p) \wedge \text{alignedI}(smt\_pval, y_p) \wedge [y_p] 1 \xrightarrow{\times}_{\tau} [y] \star I
\end{array}
\quad \text{ACTION\_CREATE}$$

$$\begin{array}{c}
\mathcal{C}; \mathcal{L}; \Phi \vdash smt\_pval_1 \Rightarrow \cdot : \text{loc}. \_ \\
\mathcal{C}; \mathcal{L}; \Phi \vdash smt\_pval' \Rightarrow \cdot' : \text{of\_ctype}(\tau). \cdot' \\
\text{smt}(\Phi \Rightarrow \text{representable}(\tau, smt\_pval')) \\
\text{smt}(\Phi \Rightarrow smt\_pval_0 = smt\_pval_1) \\
\hline
\mathcal{C}; \mathcal{L}; \Phi; \cdot, [smt\_pval_0] 1 \mapsto_{\tau} [\cdot'] \vdash \text{store}(\cdot, \tau, smt\_pval_1, smt\_pval', \cdot) \Rightarrow \Sigma \cdot : \text{unit}. [smt\_pval_0] 1 \xrightarrow{\check{\times}}_{\tau} [smt\_pval'] \star I
\end{array}
\quad \text{ACTION\_STORE}$$

$$\begin{array}{c}
\mathcal{C}; \mathcal{L}; \Phi \vdash smt\_pval_1 \Rightarrow \cdot : \text{loc}. \_ \\
\text{smt}(\Phi \Rightarrow smt\_pval_0 = smt\_pval_1) \\
\hline
\mathcal{C}; \mathcal{L}; \Phi; \cdot, [smt\_pval_0] 1 \mapsto_{\tau} [\cdot'] \vdash \text{kill}(\text{static } \tau, smt\_pval_1) \Rightarrow \Sigma \cdot : \text{unit}. I
\end{array}
\quad \text{ACTION\_KILL\_STATIC}$$

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

$$\begin{array}{c}
\mathcal{C}; \mathcal{L}; \Phi \vdash pval \Rightarrow y : \beta. term \\
\hline
\mathcal{C}; \mathcal{L}; \Phi; \cdot \vdash pval \Rightarrow \Sigma y : \beta. term \wedge I
\end{array}
\quad \text{SEQ\_EXPR\_PURE}$$

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

$$\begin{array}{c}
\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash mem\_action \Rightarrow ret \\
\hline
\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \text{Pos } mem\_action \Rightarrow ret
\end{array}
\quad \text{IS\_EXPR\_ACTION}$$

$$\frac{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \text{mem\_action} \Rightarrow \text{ret}}{\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \text{Neg mem\_action} \Rightarrow \text{ret}} \quad \text{IS\_EXPR\_NEG\_ACTION}$$

Definition rules: 39 good 0 bad  
Definition rule clauses: 90 good 0 bad