tag	OCaml type for struct/union tag
impl_const	implementation-defined constant
x, y, v, *ident*	OCaml type variable for symbols
mem_int	memory integer value
member	C struct/union member name
annots	annotations
nat	OCaml arbitrary-width natural number
n, i	index variables
loc	OCaml type for C source
mem_iv_c	OCaml type for memory constraints on integer values
UB_name	undefined behaviour
string	OCaml string
tyvar_TY	OCaml type variable for types
	OCaml type for symbol prefix
mem_order	OCaml type for memory order
linux_mem_order	OCaml type for Linux memory order
k	OCaml fixed-width integer

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

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

	|	mem_int	integer value
	|	mem_ptr	pointer value
	|	$\texttt{array}\,(\,\overline{loaded\_value_i}^{\,i}\,)$	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
$loaded\_value$	::=		potentially unspecified C object values
	|	$\texttt{specified}\,object\_value$	specified loaded value
β	::=		Ott-hack, ignore
$value$	::=		Core values
	|	$object\_value$	C object value
	|	$loaded\_value$	loaded C object value
	|	`Unit`	unit
	|	`True`	boolean true
	|	`False`	boolean false
	|	$\beta[value_1, \,..\,, value_i]$	list
	|	$(value_1, \,..\,, value_i)$	tuple
$ctor$	::=		data constructors
	|	$\texttt{Nil}\,\beta$	empty list
	|	`Cons`	list cons
	|	`Tuple`	tuple
	|	`Array`	C array
	|	`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
	|	`Specified`	non-unspecified loaded value
	|	`Fvfromint`	cast integer to floating value
	|	`Ivfromfloat`	cast floating to integer value
$\mathit{ident\_opt\_\beta}$	::=		type annotated optional identifier
	|	$\_ : \beta$	
	|	$\mathit{ident} : \beta$	
$\mathit{pattern\_aux}$	::=		
	|	$\mathit{ident\_opt\_\beta}$	
	|	$\mathit{ctor}(\overline{\mathit{pattern}_i}^{\,i})$	
$\mathit{pattern}$	::=		
	|	$\mathit{loc}\ \mathit{annots}\ \mathit{pattern\_aux}$	
$\mathit{ident\_or\_pattern}$	::=		
	|	ident	
	|	$\mathit{pattern}$	
ident	::=		Ott-hack, ignore
$\mathit{pexpr\_aux}$	::=		pure expressions
	|	ident	
	|	$\mathit{impl\_const}$	implementation-defined constant
	|	value	
	|	$\texttt{constrained}\,(\overline{\mathit{mem\_iv\_c}_i, \mathit{ident}_i}^{\,i})$	constrained value
	|	$\texttt{error}\,(\mathit{string}, \mathit{ident})$	impl-defined static error

	$\vert$	$ctor(\overline{ident_i}^{\,i})$		data constructor application
	$\vert$	$\texttt{array\_shift}\,(ident_1, \tau, ident_2)$		pointer array shift
	$\vert$	$\texttt{member\_shift}\,(ident, ident, member)$		pointer struct/union member shift
	$\vert$	$\texttt{not}\,(ident)$		boolean not
	$\vert$	$ident_1\ binop\ ident_2$		binary operations
	$\vert$	$(\,\texttt{struct}\,ident)\{\,\overline{.member_i = ident_i}^{\,i}\,\}$		C struct expression
	$\vert$	$(\,\texttt{union}\,ident)\{.member = ident\}$		C union expression
	$\vert$	$\texttt{memberof}\,(ident, member, ident)$		C struct/union member access
	$\vert$	$name(ident_1, \ldots, ident_n)$		pure function call
	$\vert$	$\texttt{assert\_undef}\,(ident, loc, UB\_name)$		
	$\vert$	$\texttt{bool\_to\_integer}\,(ident)$		
	$\vert$	$\texttt{conv\_int}\,(\tau, ident)$		
	$\vert$	$\texttt{wrapI}\,(\tau, ident)$		
$pexpr$	::=			pure expressions with location and annotations
	$\vert$	$loc\ annots\ tyvar\_TY\ pexpr\_aux$		
$tpexpr\_aux$	::=			top-level pure expressions
	$\vert$	$\texttt{undef}\,loc\ UB\_name$		undefined behaviour
	$\vert$	$\texttt{case}\,ident\,\texttt{of}\,\overline{\vert\ pattern_i \Rightarrow tpexpr_i}^{\,i}\,\texttt{end}$		pattern matching
	$\vert$	$\texttt{let}\,ident\_or\_pattern = pexpr\,\texttt{in}\,tpexpr$		pure let
	$\vert$	$\texttt{if}\,ident\,\texttt{then}\,tpexpr_1\,\texttt{else}\,tpexpr_2$		pure if
	$\vert$	$\texttt{done}\,ident$		pure done
$tpexpr$	::=			pure top-level pure expressions with location and annotations
	$\vert$	$loc\ annots\ tyvar\_TY\ tpexpr\_aux$		
	$\vert$	$[\mathcal{C}/\mathcal{C}']tpexpr$	M	
m_kill_kind	::=			
	$\vert$	$\texttt{dynamic}$		

	$\vert$	`static` τ	
bool	::=		OCaml booleans
	$\vert$	`true`	
	$\vert$	`false`	
action_aux	::=		memory actions
	$\vert$	`create` $(ident, \tau)$	
	$\vert$	`create_readonly` $(ident_1, \tau, ident_2)$	
	$\vert$	`alloc` $(ident_1, ident_2)$	
	$\vert$	`kill` $(m\_kill\_kind, ident)$	the boolean indicates whether the action is dynamic (i.e. free())
	$\vert$	`store` $(bool, \tau, ident_1, ident_2, mem\_order)$	the boolean indicates whether the store is locking
	$\vert$	`load` $(\tau, ident, mem\_order)$	
	$\vert$	`rmw` $(\tau, ident_1, ident_2, ident_3, mem\_order_1, mem\_order_2)$	
	$\vert$	`fence` (mem_order)	
	$\vert$	`cmp_exch_strong` $(\tau, ident_1, ident_2, ident_3, mem\_order_1, mem\_order_2)$	
	$\vert$	`cmp_exch_weak` $(\tau, ident_1, ident_2, ident_3, mem\_order_1, mem\_order_2)$	
	$\vert$	`linux_fence` $(linux\_mem\_order)$	
	$\vert$	`linux_load` $(\tau, ident, linux\_mem\_order)$	
	$\vert$	`linux_store` $(\tau, ident_1, ident_2, linux\_mem\_order)$	
	$\vert$	`linux_rmw` $(\tau, ident_1, ident_2, linux\_mem\_order)$	
action	::=		
	$\vert$	$loc\ action\_aux$	
memop	::=		operations involving the memory state
	$\vert$	$ident_1 == ident_2$	pointer equality comparison
	$\vert$	$ident_1 \neq ident_2$	pointer inequality comparison
	$\vert$	$ident_1 < ident_2$	pointer less-than comparison
	$\vert$	$ident_1 > ident_2$	pointer greater-than comparison

	|	$\mathit{ident}_1 \leq \mathit{ident}_2$		pointer less-than comparison
	|	$\mathit{ident}_1 \geq \mathit{ident}_2$		pointer greater-than comparison
	|	$\mathit{ident}_1 -_{\tau} \mathit{ident}_2$		pointer subtraction
	|	$\texttt{intFromPtr}\,(\tau_1, \tau_2, \mathit{ident})$		cast of pointer value to integer value
	|	$\texttt{ptrFromInt}\,(\tau_1, \tau_2, \mathit{ident})$		cast of integer value to pointer value
	|	$\texttt{ptrValidForDeref}\,(\tau, \mathit{ident})$		dereferencing validity predicate
	|	$\texttt{ptrWellAligned}\,(\tau, \mathit{ident})$		
	|	$\texttt{ptrArrayShift}\,(\mathit{ident}_1, \tau, \mathit{ident}_2)$		
	|	$\texttt{memcpy}\,(\mathit{ident}_1, \mathit{ident}_2, \mathit{ident}_3)$		
	|	$\texttt{memcmp}\,(\mathit{ident}_1, \mathit{ident}_2, \mathit{ident}_3)$		
	|	$\texttt{realloc}\,(\mathit{ident}_1, \mathit{ident}_2, \mathit{ident}_3)$		
	|	$\texttt{va\_start}\,(\mathit{ident}_1, \mathit{ident}_2)$		
	|	$\texttt{va\_copy}\,(\mathit{ident})$		
	|	$\texttt{va\_arg}\,(\mathit{ident}, \tau)$		
	|	$\texttt{va\_end}\,(\mathit{ident})$		
paction	::=			memory actions with polarity
	|	*action*	M	positive, sequenced by both `let weak` and `let strong`
	|	$\neg\,(\mathit{action})$	M	negative, only sequenced by `let strong`
expr_aux	::=			(effectful) expressions
	|	$\texttt{pure}\,(\mathit{pexpr})$		pure expression
	|	$\texttt{memop}\,(\mathit{memop})$		pointer op involving memory
	|	*paction*		memory action
	|	`skip`		skip
	|	$\texttt{ccall}\,(\tau, \mathit{ident}, \overline{\mathit{ident}_i}^{\,i})$		C function call
	|	$\texttt{pcall}\,(\mathit{name}, \overline{\mathit{ident}_i}^{\,i})$		procedure call
expr	::=			(effectful) expressions with location and annotations
	|	*loc annots expr_aux*		

$texpr\_aux$	::=		top-level expressions
	|	$\texttt{let}\ ident\_or\_pattern = pexpr\ \texttt{in}\ texpr$	
	|	$\texttt{let}\ \texttt{weak}\ pattern = expr\ \texttt{in}\ texpr$	weak sequencing
	|	$\texttt{let}\ \texttt{strong}\ ident\_or\_pattern = expr\ \texttt{in}\ texpr$	strong sequencing
	|	$\texttt{case}\ ident\ \texttt{with}\ \overline{\mid pattern_i \Rightarrow texpr_i}^{i}\ \texttt{end}$	pattern matching
	|	$\texttt{if}\ ident\ \texttt{then}\ texpr_1\ \texttt{else}\ texpr_2$	conditional
	|	$\texttt{bound}\,[k](texpr)$	??, doesn't exist at runtime
	|	$\texttt{unseq}\,(expr_1, \,..\, , expr_n)$	unsequenced expressions
	|	$\texttt{nd}\,(texpr_1, \,..\, , texpr_n)$	nondeterministic sequencing
	|	$\texttt{done}\ ident$	end of top-level expression
	|	$\texttt{undef}\ loc\ UB\_name$	undefined behaviour
	|	$\texttt{run}\ ident\ ident_1, \,..\, , ident_n$	run from label
$texpr$	::=		top-level expressions with location and annotations
	|	$loc\ annots\ texpr\_aux$	
$terminals$	::=		
	|	λ	
	|	$\longrightarrow$	
	|	$\rightarrow$	
	|	$\leadsto$	
	|	$\Rightarrow$	
	|	$\Leftarrow$	
	|	$\vdash$	
	|	$\in$	
	|	Π	
	|	$\forall$	
	|	$\multimap$	
	|	$\supset$	
	|	Σ	

	$\mid$	$\exists$		
	$\mid$	$\star$		
	$\mid$	$\times$		
	$\mid$	$\wedge$		
	$\mid$	$\bigwedge$		
	$\mid$	$\neg$		
	$\mid$	$=$		
	$\mid$	$\neq$		
	$\mid$	$\leq$		
	$\mid$	$\geq$		
	$\mid$	$\&$		
	$\mid$	$\cdot$		
	$\mid$	$\mid$		
	$\mid$	$+_{\mathrm{ptr}}$		
	$\mid$	$\mapsto$		
	$\mid$	$*$		
z	$::=$			OCaml arbitrary-width integer
	$\mid$	of_mem_int(*mem_int*)	M	
	$\mid$	of_nat(*nat*)	M	
	$\mid$	of_ctype(τ)	M	size of a C type
	$\mid$	`ptr_size`	M	size of a pointer
$\mathbb{Q}$	$::=$			Ott-hack, ignore
	$\mid$	$\frac{k_1}{k_2}$		
lit	$::=$			
	$\mid$	*ident*		
	$\mid$	()		
	$\mid$	*bool*		

$$
\begin{array}{lll}
 & | & \mathtt{int}\, z \\
 & | & \mathbb{Q} \\
 & | & \mathtt{ptr}\, z \\
 & & \\
\mathit{bool\_op} & ::= & \\
 & | & \neg\, \mathit{term} \\
 & | & \mathit{term}_1 = \mathit{term}_2 \\
 & | & \bigwedge(\, \overline{\mathit{term}_i}^{\,i}\,) \\
 & & \\
\mathit{arith\_op} & ::= & \\
 & | & \mathit{term}_1 \times \mathit{term}_2 \\
 & & \\
\mathit{list\_op} & ::= & \\
 & | & \mathtt{nil} \\
 & | & \mathit{term}_1 :: \mathit{term}_2 \\
 & | & [\mathit{term}_1, \, .. \, , \mathit{term}_n] \\
 & | & \mathit{term}^{(k)} \\
 & & \\
\mathit{tuple\_op} & ::= & \\
 & | & (\mathit{term}_1, \, .. \, , \mathit{term}_n) \\
 & | & \mathit{term}^{(k)} \\
 & & \\
\mathit{pointer\_op} & ::= & \\
 & | & \mathtt{nullop} \\
 & | & \mathit{term}_1 +_{\mathrm{ptr}} \mathit{term}_2 \\
 & & \\
\mathit{option\_op} & ::= & \\
 & | & \mathtt{none}\, \mathit{BT\_t} \\
 & | & \mathtt{some}\, \mathit{term}
\end{array}
$$

$array\_op$	$::=$		
	$\mid$	$term[\,\texttt{int}\,z]$	
$param\_op$	$::=$		
	$\mid$	$term(term_1, \,..\,, term_n)$	
$struct\_op$	$::=$		
	$\mid$	$term.member$	
ct_pred	$::=$		
	$\mid$	$\texttt{representable}\,(\tau, term)$	
	$\mid$	$\texttt{alignedI}\,(term_1, term_2)$	
$term\_aux$	$::=$		
	$\mid$	$arith\_op$	
	$\mid$	$bool\_op$	
	$\mid$	$tuple\_op$	
	$\mid$	$struct\_op$	
	$\mid$	$pointer\_op$	
	$\mid$	$list\_op$	
	$\mid$	$array\_op$	
	$\mid$	ct_pred	
	$\mid$	$option\_op$	
	$\mid$	$param\_op$	
bt	$::=$		Ott-hack, ignore
BT_t	$::=$		Ott-hack, ignore
$term$	$::=$		

	|	*lit*		
	|	$term\_aux\,bt$		
	|	$(term)$	S	parentheses
	|	$[term_1/ident]term_2$	M	
IT_t	::=			Ott-hack, ignore
IT_t_list	::=			Ott-hack, ignore
point	::=			points-to predicate
	|	$IT\_t_1 \mapsto_{z, IT\_t_2} IT\_t_3$		
$predicate\_name$	::=			names of predicates
	|	$Sctypes\_t$		
	|	*string*		
predicate	::=			arbitrary predicate
	|	$predicate\_name(IT\_t_1, IT\_t\_list_1 \mapsto IT\_t\_list_2)$		
resource	::=			
	|	*point*		
	|	*predicate*		
arg	::=			argument types
	|	$\Pi\, ident : \beta . arg$		
	|	$\forall\, ident : \beta . arg$		
	|	$resource \multimap arg$		
	|	$term \supset arg$		
	|	$\mathtt{I}$		

ret	::=			return types
	|	$\Sigma\, ident : \beta.ret$		
	|	$\exists\, ident : \beta.ret$		
	|	$resource \star ret$		
	|	$term \wedge ret$		
	|	$\mathtt{I}$		
$\mathcal{C}$	::=			computational var env
	|	$.$		
	|	$\mathcal{C}, ident : BT\_t$		
	|	$\mathcal{C}, \mathcal{C}'$		
	|	$\mathrm{fresh}(\mathcal{C})$	M	identical context except with fresh variable names
$\mathcal{L}$	::=			logical var env
	|	$.$		
	|	$\mathcal{L}, ident$		
Φ	::=			constraints env
	|	$.$		
	|	$\Phi, term$		
	|	Φ, ret		Ott-hack, ignore
$\mathcal{R}$	::=			resources env
	|	$.$		
	|	$\mathcal{R}, resource$		
$formula$	::=			
	|	$judgement$		
	|	$\mathtt{smt}\,(\Phi \Rightarrow ret)$		theorem to be proved: ret only consists of logical constraints
	|	$\overline{ident_i : \beta_i \in \mathcal{C}_i}^{\,i}$		

	|	$\mathit{ident} : \texttt{struct}\, \mathit{tag}\ \&\ \overline{\mathit{member}_i : \tau_i}^{\,i} \in \texttt{Globals}$
	|	$\overline{\mathcal{C}_i; \mathcal{L}_i; \Phi_i \vdash \mathit{mem\_val}_i \Rightarrow \texttt{mem}\, y_i, \beta_i, \mathit{term}_i}^{\,i}$
	|	$\overline{\mathcal{C}_i; \mathcal{L}_i; \Phi_i \vdash \mathit{value}_i \Rightarrow y_i, \beta_i, \mathit{term}_i}^{\,i}$
	|	$\overline{\mathit{pattern}_i : \beta_i \rightsquigarrow \mathcal{C}_i}^{\,i}$
	|	$\overline{\mathcal{C}_i; \mathcal{L}_i; \Phi_i \vdash \mathit{tpexpr}_i \Leftarrow \mathit{ret}_i}^{\,i}$
mem_value_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{mem\_val} \Rightarrow \texttt{mem}\, y, \beta, \mathit{term}$
value_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{object\_value} \Rightarrow \texttt{obj}\, \mathit{ident}, \beta, \mathit{term}$
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{value} \Rightarrow \mathit{ident}, \beta, \mathit{term}$
pexpr_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pexpr\_aux} \Rightarrow \mathit{ret}$
pattern_jtype	::=	
	|	$\mathit{pattern} : \beta \rightsquigarrow \mathcal{C}$
	|	$\mathit{ident\_or\_pattern} : \beta \rightsquigarrow \mathcal{C}$
tpexpr_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{pexpr} \Rightarrow \mathit{ret}$
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{tpexpr} \Leftarrow \mathit{ret}$
	|	$\mathcal{C}; \mathcal{L}; \Phi \vdash \mathit{tpexpr\_aux} \Leftarrow \mathit{ret}$
expr_jtype	::=	
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{action\_aux} \Rightarrow \mathit{ret}$
	|	$\mathcal{C}; \mathcal{L}; \Phi; \mathcal{R} \vdash \mathit{expr\_aux} \Rightarrow \mathit{ret}$
judgement	::=	

$$
\begin{array}{lll}
 & | & mem\_value\_jtype \\
 & | & value\_jtype \\
 & | & pexpr\_jtype \\
 & | & pattern\_jtype \\
 & | & tpexpr\_jtype \\
 & | & expr\_jtype \\
 & & \\
user\_syntax & ::= & \\
 & | & tag \\
 & | & impl\_const \\
 & | & x \\
 & | & mem\_int \\
 & | & member \\
 & | & annots \\
 & | & nat \\
 & | & n \\
 & | & loc \\
 & | & mem\_iv\_c \\
 & | & UB\_name \\
 & | & string \\
 & | & tyvar\_TY \\
 & | & \\
 & | & mem\_order \\
 & | & linux\_mem\_order \\
 & | & k \\
 & | & Sctypes\_t \\
 & | & \tau \\
 & | & \beta \\
 & | & binop \\
 & | & polarity
\end{array}
$$

| $ident$
| $name$
| τ
| mem_ptr
| mem_val
| $object\_value$
| $loaded\_value$
| β
| $value$
| $ctor$
| $ident\_opt\_\beta$
| $pattern\_aux$
| $pattern$
| $ident\_or\_pattern$
| $ident$
| $pexpr\_aux$
| $pexpr$
| $tpexpr\_aux$
| $tpexpr$
| m_kill_kind
| $bool$
| $action\_aux$
| $action$
| $memop$
| $paction$
| $expr\_aux$
| $expr$
| $texpr\_aux$
| $texpr$

$$
\begin{array}{ll}
| & \mathit{terminals} \\
| & z \\
| & \mathbb{Q} \\
| & \mathit{lit} \\
| & \mathit{bool\_op} \\
| & \mathit{arith\_op} \\
| & \mathit{list\_op} \\
| & \mathit{tuple\_op} \\
| & \mathit{pointer\_op} \\
| & \mathit{option\_op} \\
| & \mathit{array\_op} \\
| & \mathit{param\_op} \\
| & \mathit{struct\_op} \\
| & \mathit{ct\_pred} \\
| & \mathit{term\_aux} \\
| & \mathit{bt} \\
| & \mathit{BT\_t} \\
| & \mathit{term} \\
| & \mathit{IT\_t} \\
| & \mathit{IT\_t\_list} \\
| & \mathit{point} \\
| & \mathit{predicate\_name} \\
| & \mathit{predicate} \\
| & \mathit{resource} \\
| & \mathit{arg} \\
| & \mathit{ret} \\
| & \mathcal{C} \\
| & \mathcal{L} \\
| & \Phi
\end{array}
$$

$\mid \quad \mathcal{R}$
$\mid \quad \mathit{formula}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{object\_value} \Rightarrow \texttt{obj}\, y, \beta, \mathit{term}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathit{object\_value} \Rightarrow y, \beta, \mathit{term}} \quad \textsc{Val\_Obj}$$

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

$$\frac{}{\mathcal{C};\mathcal{L};\Phi \vdash \texttt{Unit} \Rightarrow y, \texttt{unit}, y = ()} \quad \textsc{Val\_Unit}$$

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

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

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash value_1 \Rightarrow y_1, \beta, term_1 \,..\, \mathcal{C};\mathcal{L};\Phi \vdash value_n \Rightarrow y_n, \beta, term_n}{\mathcal{C};\mathcal{L};\Phi \vdash \beta[value_1, \,..\,, value_n] \Rightarrow y, [\beta], \bigwedge([y^{(k_1)}/y_1]term_1, \,..\,, [y^{(k_n)}/y_n]term_n)} \quad \textsc{Val\_List}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash value_1 \Rightarrow y_1, \beta_1, term_1 \,..\, \mathcal{C};\mathcal{L};\Phi \vdash value_n \Rightarrow y_n, \beta_n, term_n}{\mathcal{C};\mathcal{L};\Phi \vdash (value_1, \,..\,, value_n) \Rightarrow y, (\beta_1, \,..\,, \beta_n), \bigwedge([y^{(k_1)}/y_1]term_1, \,..\,, [y^{(k_n)}/y_n]term_n)} \quad \textsc{Val\_Tuple}$$

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

$$\frac{x : \beta \in \mathcal{C}}{\mathcal{C};\mathcal{L};\Phi \vdash x \Rightarrow \Sigma\, y : \beta. y = x \wedge \mathtt{I}} \quad \textsc{PExpr\_Aux\_Var}$$

$$\frac{\mathcal{C};\mathcal{L};\Phi \vdash value \Rightarrow y, \beta, term}{\mathcal{C};\mathcal{L};\Phi \vdash value \Rightarrow \Sigma\, y : \beta. term \wedge \mathtt{I}} \quad \textsc{PExpr\_Aux\_Val}$$

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

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

$$\frac{\begin{array}{c} x_1 : \beta \in \mathcal{C} \\ x_2 : [\beta] \in \mathcal{C} \end{array}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathtt{Cons}(x_1, x_2) \Rightarrow \Sigma\, y : [\beta]. y = x_1 :: x_2 \wedge \mathtt{I}} \quad \textsc{PExpr\_Aux\_Ctor\_Cons}$$

$$\frac{x_1 : \beta_1 \in \mathcal{C} \,..\, x_n : \beta_n \in \mathcal{C}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathtt{Tuple}(x_1, \,..\,, x_n) \Rightarrow \Sigma\, y : (\beta_1, \,..\,, \beta_n). y = (x_1, \,..\,, x_n) \wedge \mathtt{I}} \quad \textsc{PExpr\_Aux\_Ctor\_Tuple}$$

$$\frac{x_1 : \beta \in \mathcal{C} \,..\, x_n : \beta \in \mathcal{C}}{\mathcal{C};\mathcal{L};\Phi \vdash \mathtt{Array}(x_1, \,..\,, x_n) \Rightarrow \Sigma\, y : \mathtt{array}\,\beta. \bigwedge(y[\mathtt{int}\, z_1] = x_1, \,..\,, y[\mathtt{int}\, z_n] = x_n) \wedge \mathtt{I}} \quad \textsc{PExpr\_Aux\_Ctor\_Array}$$

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

$$\frac{\begin{array}{l} x_1 : \texttt{loc} \in \mathcal{C} \\ x_2 : \texttt{integer} \in \mathcal{C} \end{array}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{array\_shift}\,(x_1, \tau, x_2) \Rightarrow \Sigma\, y : \texttt{loc}. y = x_1 +_{\text{ptr}} x_2 \times \texttt{int}\,\text{of\_ctype}(\tau) \ \wedge \texttt{I}} \quad \textsc{PExpr\_Aux\_Array\_Shift}$$

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

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

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

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

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

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

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

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

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

$$\frac{\begin{array}{l} x : \texttt{bool} \in \mathcal{C} \\ \mathcal{C}; \mathcal{L}; \Phi, x = \texttt{true} \vdash \mathit{tpexpr}_1 \Leftarrow \mathit{ret} \\ \mathcal{C}; \mathcal{L}; \Phi, x = \texttt{false} \vdash \mathit{tpexpr}_2 \Leftarrow \mathit{ret} \end{array}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{if}\ x\ \texttt{then}\ \mathit{tpexpr}_1\ \texttt{else}\ \mathit{tpexpr}_2 \Leftarrow \mathit{ret}} \quad \textsc{TPExpr\_Aux\_If}$$

$$\frac{\begin{array}{c} x : \beta \in \mathcal{C} \\ \texttt{smt}\,(\Phi \Rightarrow \mathit{ret}) \end{array}}{\mathcal{C}; \mathcal{L}; \Phi \vdash \texttt{done}\, x \Leftarrow \Sigma\, y : \beta.\mathit{ret}} \quad \textsc{TPExpr\_Aux\_Done}$$

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

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

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

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

$$\frac{\begin{array}{c} x_1 : \texttt{loc} \in \mathcal{C} \\ x_2 : \mathrm{of\_ctype}(\tau) \in \mathcal{C} \end{array}}{\mathcal{C}; \mathcal{L}; \Phi; \cdot, x_1 \mapsto_{\texttt{ptr\_size},1} \mathit{term} \vdash \texttt{store}\,(\mathit{bool}, \tau, x_1, x_2, \mathit{mem\_order}) \Rightarrow \Sigma\, y : \texttt{unit}.x_1 \mapsto_{\texttt{ptr\_size},1} x_2 \star \texttt{I}} \quad \textsc{Action\_Store}$$

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

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

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

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

```
Definition rules:        41 good    0 bad
Definition rule clauses: 80 good    0 bad
```