tag	OCaml type for struct/union tag
$impl\_const$	implementation-defined constant
x, y, $ident$	OCaml type variable for symbols
ty_mem_int	memory integer value
mem_val	memory value
$member$	C struct/union member name
τ	C type
$annots$	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
τ	OCaml type for an annotated C type
sym_prefix	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

β	::=		Core base types
	|	`unit`	unit
	|	`bool`	boolean
	|	`integer`	integer
	|	`real`	rational numbers?
	|	`loc`	location
	|	$[\beta]$	list
	|	$(\beta_1, .., \beta_n)$	tuple
	|	`struct` *tag*	struct
	|	$\{\beta\}$	set
	|	`opt` (β)	option
	|	$\beta_1, .., \beta_n \to \beta$	parameter types
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

	$\mid$	`Pos`	sequenced by `let weak` and `let strong`
	$\mid$	`Neg`	only sequenced by `let strong`
ident	$::=$		Ott-hack, ignore
	$\mid$	ident	
name	$::=$		
	$\mid$	ident	Core identifier
	$\mid$	$\mathit{impl\_const}$	implementation-defined constant
ptrval	$::=$		pointer values
	$\mid$	$\mathtt{nullptr}$	null pointer
	$\mid$	$\mathtt{funcptr}\,\mathit{ident}$	function pointer
	$\mid$	$\mathtt{concptr}\,\mathit{nat}$	concrete pointer
$\mathit{object\_value}$	$::=$		C object values
	$\mid$	$\mathit{ty\_mem\_int}$	integer value
	$\mid$	ptrval	pointer value
	$\mid$	$\mathtt{array}\,(\mathit{loaded\_value}_1, \,..\,, \mathit{loaded\_value}_n)$	C array value
	$\mid$	$(\mathtt{struct}\,\mathit{ident})\{\,\overline{.\mathit{member}_i : \tau_i = \mathit{mem\_val}_i}^{\,i}\,\}$	C struct value
	$\mid$	$(\mathtt{union}\,\mathit{ident})\{.\mathit{member} = \mathit{mem\_val}\}$	C union value
$\mathit{loaded\_value}$	$::=$		potentially unspecified C object values
	$\mid$	$\mathtt{specified}\,(\mathit{object\_value})$	specified loaded value
β	$::=$		Ott-hack, ignore
	$\mid$	β	
value	$::=$		Core values
	$\mid$	$\mathit{object\_value}$	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
	|	`Nil` β	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
	|	`Unspecified`	unspecified loaded value
	|	`Fvfromint`	cast integer to floating value
	|	`Ivfromfloat`	cast floating to integer value
$ident\_opt\_\beta$	::=		type annotated optional identifier
	|	$\_ : \beta$	
	|	$ident : \beta$	
$pattern\_aux$	::=		

	$\mid$	$ident\_opt\_\beta$	
	$\mid$	$ctor(\,\overline{pattern_i}^{\,i}\,)$	
$pattern$	$::=$		
	$\mid$	$loc\ annots\ pattern\_aux$	
$ident\_or\_pattern$	$::=$		
	$\mid$	$ident$	
	$\mid$	$pattern$	
$ident$	$::=$		Ott-hack, ignore
	$\mid$	$ident$	
$pexpr\_aux$	$::=$		pure expressions
	$\mid$	$ident$	
	$\mid$	$impl\_const$	implementation-defined constant
	$\mid$	$value$	
	$\mid$	$\texttt{constrained}\,(\,\overline{mem\_iv\_c_i, ident_i}^{\,i}\,)$	constrained value
	$\mid$	$\texttt{error}\,(string, ident)$	impl-defined static error
	$\mid$	$ctor(\,\overline{ident_i}^{\,i}\,)$	data constructor application
	$\mid$	$\texttt{array\_shift}\,(ident_1, \tau, ident_2)$	pointer array shift
	$\mid$	$\texttt{member\_shift}\,(ident, ident, member)$	pointer struct/union member shift
	$\mid$	$\texttt{not}\,(ident)$	boolean not
	$\mid$	$ident_1\ binop\ ident_2$	binary operations
	$\mid$	$(\,\texttt{struct}\ ident)\{\,\overline{.member_i = ident_i}^{\,i}\,\}$	C struct expression
	$\mid$	$(\,\texttt{union}\ ident)\{.member = ident\}$	C union expression
	$\mid$	$\texttt{memberof}\,(ident, member, ident)$	C struct/union member access
	$\mid$	$name(ident_1, \,..\, , ident_n)$	pure function call
	$\mid$	$\texttt{assert\_undef}\,(ident, loc, UB\_name)$	
	$\mid$	$\texttt{bool\_to\_integer}\,(ident)$	

	|	$\texttt{conv\_int}\,(\tau, \mathit{ident})$	
	|	$\texttt{wrapI}\,(\tau, \mathit{ident})$	
pexpr	::=		pure expressions with location and annotations
	|	$\mathit{loc}\ \mathit{annots}\ \mathit{tyvar\_TY}\ \mathit{pexpr\_aux}$	
$\mathit{tpexpr\_aux}$	::=		top-level pure expressions
	|	$\texttt{undef}\,\mathit{loc}\ \mathit{UB\_name}$	undefined behaviour
	|	$\texttt{case}\,\mathit{ident}\,\texttt{of}\,\overline{\lvert \mathit{pattern}_i \Rightarrow \mathit{tpexpr}_i}^{\,i}\,\texttt{end}$	pattern matching
	|	$\texttt{let}\,\mathit{ident\_or\_pattern} = \mathit{tpexpr}_1\,\texttt{in}\,\mathit{tpexpr}_2$	pure let
	|	$\texttt{if}\,\mathit{ident}\,\texttt{then}\,\mathit{tpexpr}_1\,\texttt{else}\,\mathit{tpexpr}_2$	pure if
	|	$\texttt{done}\,\mathit{ident}$	pure done
tpexpr	::=		pure top-level pure expressions with location and annotations
	|	$\mathit{loc}\ \mathit{annots}\ \mathit{tyvar\_TY}\ \mathit{tpexpr\_aux}$	
$\mathit{m\_kill\_kind}$	::=		
	|	$\texttt{dynamic}$	
	|	$\texttt{static}\,\tau$	
bool	::=		OCaml booleans
	|	$\texttt{true}$	
	|	$\texttt{false}$	
$\mathit{action\_aux}$	::=		memory actions
	|	$\texttt{create}\,(\mathit{ident}, \tau)\mathit{sym\_prefix}$	
	|	$\texttt{create\_readonly}\,(\mathit{ident}_1, \tau, \mathit{ident}_2)\mathit{sym\_prefix}$	
	|	$\texttt{alloc}\,(\mathit{ident}_1, \mathit{ident}_2)\mathit{sym\_prefix}$	
	|	$\texttt{kill}\,(\mathit{m\_kill\_kind}, \mathit{ident})$	the boolean indicates whether the action is dynamic (i.e. free())
	|	$\texttt{store}\,(\mathit{bool}, \tau, \mathit{ident}_1, \mathit{ident}_2, \mathit{mem\_order})$	the boolean indicates whether the store is locking

	$\mid$	$\texttt{load}\,(\tau, \mathit{ident}, \mathit{mem\_order})$	
	$\mid$	$\texttt{rmw}\,(\tau, \mathit{ident}_1, \mathit{ident}_2, \mathit{ident}_3, \mathit{mem\_order}_1, \mathit{mem\_order}_2)$	
	$\mid$	$\texttt{fence}\,(\mathit{mem\_order})$	
	$\mid$	$\texttt{cmp\_exch\_strong}\,(\tau, \mathit{ident}_1, \mathit{ident}_2, \mathit{ident}_3, \mathit{mem\_order}_1, \mathit{mem\_order}_2)$	
	$\mid$	$\texttt{cmp\_exch\_weak}\,(\tau, \mathit{ident}_1, \mathit{ident}_2, \mathit{ident}_3, \mathit{mem\_order}_1, \mathit{mem\_order}_2)$	
	$\mid$	$\texttt{linux\_fence}\,(\mathit{linux\_mem\_order})$	
	$\mid$	$\texttt{linux\_load}\,(\tau, \mathit{ident}, \mathit{linux\_mem\_order})$	
	$\mid$	$\texttt{linux\_store}\,(\tau, \mathit{ident}_1, \mathit{ident}_2, \mathit{linux\_mem\_order})$	
	$\mid$	$\texttt{linux\_rmw}\,(\tau, \mathit{ident}_1, \mathit{ident}_2, \mathit{linux\_mem\_order})$	
action	$::=$		
	$\mid$	$\mathit{loc}\,\mathit{action\_aux}$	
memop	$::=$		operations involving the memory state
	$\mid$	$\mathit{ident}_1 == \mathit{ident}_2$	pointer equality comparison
	$\mid$	$\mathit{ident}_1 \neq \mathit{ident}_2$	pointer inequality comparison
	$\mid$	$\mathit{ident}_1 < \mathit{ident}_2$	pointer less-than comparison
	$\mid$	$\mathit{ident}_1 > \mathit{ident}_2$	pointer greater-than comparison
	$\mid$	$\mathit{ident}_1 \leq \mathit{ident}_2$	pointer less-than comparison
	$\mid$	$\mathit{ident}_1 \geq \mathit{ident}_2$	pointer greater-than comparison
	$\mid$	$\mathit{ident}_1 -_\tau \mathit{ident}_2$	pointer subtraction
	$\mid$	$\texttt{intFromPtr}\,(\tau_1, \tau_2, \mathit{ident})$	cast of pointer value to integer value
	$\mid$	$\texttt{ptrFromInt}\,(\tau_1, \tau_2, \mathit{ident})$	cast of integer value to pointer value
	$\mid$	$\texttt{ptrValidForDeref}\,(\tau, \mathit{ident})$	dereferencing validity predicate
	$\mid$	$\texttt{ptrWellAligned}\,(\tau, \mathit{ident})$	
	$\mid$	$\texttt{ptrArrayShift}\,(\mathit{ident}_1, \tau, \mathit{ident}_2)$	
	$\mid$	$\texttt{memcpy}\,(\mathit{ident}_1, \mathit{ident}_2, \mathit{ident}_3)$	
	$\mid$	$\texttt{memcmp}\,(\mathit{ident}_1, \mathit{ident}_2, \mathit{ident}_3)$	
	$\mid$	$\texttt{realloc}\,(\mathit{ident}_1, \mathit{ident}_2, \mathit{ident}_3)$	TODO: not sure about this
	$\mid$	$\texttt{va\_start}\,(\mathit{ident}_1, \mathit{ident}_2)$	

	|	$\texttt{va\_copy}\,(\mathit{ident})$		
	|	$\texttt{va\_arg}\,(\mathit{ident}, \tau)$		
	|	$\texttt{va\_end}\,(\mathit{ident})$		
paction	::=			memory actions with polarity
	|	*polarity action*		
	|	*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
	|	$\texttt{pure}\,(\mathit{pexpr})$		
	|	$\texttt{memop}\,(\mathit{memop})$		pointer op involving memory
	|	*paction*		memory action
	|	`skip`		
	|	$\texttt{ccall}\,(\tau, \mathit{ident}, \overline{\mathit{ident}_i}^{i})$		C function call
	|	$\texttt{pcall}\,(\mathit{name}, \overline{\mathit{ident}_i}^{i})$		Core procedure call
expr	::=			(effectful) expressions with location and annotations
	|	*loc annots expr_aux*		
texpr_aux	::=			top-level expressions
	|	$\texttt{let}\,\mathit{ident\_or\_pattern} = \mathit{pexpr}\,\texttt{in}\,\mathit{texpr}$		
	|	$\texttt{let}\,\texttt{weak}\,\mathit{pattern} = \mathit{expr}\,\texttt{in}\,\mathit{texpr}$		weak sequencing
	|	$\texttt{let}\,\texttt{strong}\,\mathit{ident\_or\_pattern} = \mathit{expr}\,\texttt{in}\,\mathit{texpr}$		strong sequencing
	|	$\texttt{case}\,\mathit{ident}\,\texttt{with}\,\overline{\mid \mathit{pattern}_i \Rightarrow \mathit{texpr}_i}^{i}\,\texttt{end}$		pattern matching
	|	$\texttt{if}\,\mathit{ident}\,\texttt{then}\,\mathit{texpr}_1\,\texttt{else}\,\mathit{texpr}_2$		
	|	$\texttt{bound}\,[k](\mathit{texpr})$		...and boundary
	|	$\texttt{unseq}\,(\mathit{expr}_1, \,..\,, \mathit{expr}_n)$		unsequenced expressions
	|	$\texttt{nd}\,(\mathit{texpr}_1, \,..\,, \mathit{texpr}_n)$		nondeterministic sequencing
	|	$\texttt{done}\,\mathit{ident}$		

	$\|$	**undef** loc UB_name	
	$\|$	**run** $ident$ $ident_1, .., ident_n$	run from label
$texpr$	$::=$		top-level expressions with location and annotations
	$\|$	loc $annots$ $texpr\_aux$	
$terminals$	$::=$		
	$\|$	λ	
	$\|$	$\longrightarrow$	
	$\|$	$\rightarrow$	
	$\|$	$\Rightarrow$	
	$\|$	$\Leftarrow$	
	$\|$	$\vdash$	
	$\|$	$\in$	
	$\|$	Π	
	$\|$	$\forall$	
	$\|$	$\multimap$	
	$\|$	$\supset$	
	$\|$	Σ	
	$\|$	$\exists$	
	$\|$	$\star$	
	$\|$	$\wedge$	
	$\|$	$\bigwedge$	
	$\|$	$\neg$	
	$\|$	$=$	
	$\|$	$\neq$	
	$\|$	$\leq$	
	$\|$	$\geq$	
z	$::=$		OCaml arbitrary-width integer

	$\vert$	$\texttt{of\_mem\_int}\ ty\_mem\_int$	M
	$\vert$	$\texttt{of\_nat}\ nat$	M
lit	$::=$		
	$\vert$	$ident$	
	$\vert$	$()$	
	$\vert$	$bool$	
	$\vert$	$\texttt{int}\ z$	
	$\vert$	$\texttt{ptr}\ z$	
$bool\_op$	$::=$		
	$\vert$	$\neg\ index\_term$	
	$\vert$	$index\_term_1 = index\_term_2$	
	$\vert$	$\bigwedge(index\_term_1, \ldots, index\_term_n)$	
$list\_op$	$::=$		
	$\vert$	$[index\_term_1, \ldots, index\_term_n]$	
	$\vert$	$index\_term^{(k)}$	
$tuple\_op$	$::=$		
	$\vert$	$(index\_term_1, \ldots, index\_term_n)$	
	$\vert$	$index\_term^{(k)}$	
$pointer\_op$	$::=$		
	$\vert$	$\texttt{nullop}$	
$param\_op$	$::=$		
	$\vert$	$index\_term(index\_term_1, \ldots, index\_term_n)$	
$index\_term\_aux$	$::=$		

	$\vert$	$bool\_op$		
	$\vert$	$list\_op$		
	$\vert$	$pointer\_op$		
	$\vert$	$param\_op$		
bt	$::=$			OCaml type variable for base types
	$\vert$			Ott-hack, ignore
$index\_term$	$::=$			
	$\vert$	lit		
	$\vert$	$index\_term\_aux\, bt$		
	$\vert$	$(index\_term)$	S	parentheses
	$\vert$	$index\_term[index\_term_1/ident_1, .., index\_term_n/ident_n]$	M	
arg	$::=$			argument types
	$\vert$	$\Pi\, ident : \beta.arg$		
	$\vert$	$\forall\, ident : \texttt{logSort}\, .arg$		
	$\vert$	$\texttt{resource} \multimap arg$		
	$\vert$	$index\_term \supset arg$		
	$\vert$	$\texttt{I}$		
ret	$::=$			return types
	$\vert$	$\Sigma\, ident : \beta.ret$		
	$\vert$	$\exists\, ident : \texttt{logSort}\, .ret$		
	$\vert$	$\texttt{resource} \star ret$		
	$\vert$	$index\_term \wedge ret$		
	$\vert$	$\texttt{I}$		
Γ	$::=$			computational var env
	$\vert$	$\texttt{empty}$		

	$\vert$	$\Gamma, x : \beta$	
Λ	::=		logical var env
	$\vert$	`empty`	
	$\vert$	Λ, x	
Ξ	::=		constraints env
	$\vert$	`empty`	
	$\vert$	Ξ, `phi`	
formula	::=		
	$\vert$	*judgement*	
	$\vert$	`not` (*formula*)	
	$\vert$	$\mathit{ident} : \beta \ \in \ \Gamma$	
	$\vert$	$\mathit{formula}_1 \quad .. \quad \mathit{formula}_n$	
Jtype	::=		
	$\vert$	$\Gamma; \Lambda; \Xi \vdash \mathit{value} : \mathit{ident}, \beta, \mathit{index\_term}$	
	$\vert$	$\Gamma; \Lambda; \Xi \vdash \mathit{pexpr\_aux} : \mathit{ret}$	
judgement	::=		
	$\vert$	*Jtype*	
user_syntax	::=		
	$\vert$	*tag*	
	$\vert$	*impl_const*	
	$\vert$	x	
	$\vert$	*ty_mem_int*	
	$\vert$	*mem_val*	
	$\vert$	*member*	

$$
\begin{array}{ll}
| & \tau \\
| & annots \\
| & nat \\
| & n \\
| & loc \\
| & mem\_iv\_c \\
| & UB\_name \\
| & string \\
| & tyvar\_TY \\
| & \tau \\
| & sym\_prefix \\
| & mem\_order \\
| & linux\_mem\_order \\
| & k \\
| & \beta \\
| & binop \\
| & polarity \\
| & ident \\
| & name \\
| & ptrval \\
| & object\_value \\
| & loaded\_value \\
| & \beta \\
| & value \\
| & ctor \\
| & ident\_opt\_\beta \\
| & pattern\_aux \\
| & pattern \\
| & ident\_or\_pattern
\end{array}
$$

$|\quad \mathit{ident}$
$|\quad \mathit{pexpr\_aux}$
$|\quad \mathit{pexpr}$
$|\quad \mathit{tpexpr\_aux}$
$|\quad \mathit{tpexpr}$
$|\quad \mathit{m\_kill\_kind}$
$|\quad \mathit{bool}$
$|\quad \mathit{action\_aux}$
$|\quad \mathit{action}$
$|\quad \mathit{memop}$
$|\quad \mathit{paction}$
$|\quad \mathit{expr\_aux}$
$|\quad \mathit{expr}$
$|\quad \mathit{texpr\_aux}$
$|\quad \mathit{texpr}$
$|\quad \mathit{terminals}$
$|\quad z$
$|\quad \mathit{lit}$
$|\quad \mathit{bool\_op}$
$|\quad \mathit{list\_op}$
$|\quad \mathit{tuple\_op}$
$|\quad \mathit{pointer\_op}$
$|\quad \mathit{param\_op}$
$|\quad \mathit{index\_term\_aux}$
$|\quad \mathit{bt}$
$|\quad \mathit{index\_term}$
$|\quad \mathit{arg}$
$|\quad \mathit{ret}$
$|\quad \Gamma$

$|\quad \Lambda$
$|\quad \Xi$
$|\quad \mathit{formula}$

$\boxed{\Gamma; \Lambda; \Xi \vdash \mathit{value} : \mathit{ident}, \beta, \mathit{index\_term}}$

$$\frac{}{\Gamma; \Lambda; \Xi \vdash \mathit{ty\_mem\_int} : y, \mathtt{integer}, y = \mathtt{int\,of\_mem\_int}\ \mathit{ty\_mem\_int}} \quad \textsc{Val\_Obj\_Int}$$

$$\frac{}{\Gamma; \Lambda; \Xi \vdash \mathtt{nullptr} : y, \mathtt{loc}, y = \mathtt{nullop}} \quad \textsc{Val\_Obj\_Ptr\_Null}$$

$$\frac{}{\Gamma; \Lambda; \Xi \vdash \mathtt{funcptr}\, \mathit{ident} : y, \mathtt{loc}, y = \mathit{ident}} \quad \textsc{Val\_Obj\_Ptr\_Func}$$

$$\frac{}{\Gamma; \Lambda; \Xi \vdash \mathtt{concptr}\, \mathit{nat} : y, \mathtt{loc}, y = \mathtt{ptr\,of\_nat}\, \mathit{nat}} \quad \textsc{Val\_Obj\_Ptr\_Conc}$$

$$\frac{\Gamma; \Lambda; \Xi \vdash \mathit{loaded\_value}_1 : y_1, \beta, \mathit{index\_term}_1 \quad .. \quad \Gamma; \Lambda; \Xi \vdash \mathit{loaded\_value}_n : y_n, \beta, \mathit{index\_term}_n}{\Gamma; \Lambda; \Xi \vdash \mathtt{array}\,(\mathit{loaded\_value}_1, .., \mathit{loaded\_value}_n) : y, \mathtt{integer} \to \beta, \bigwedge(\mathit{index\_term}_1, .., \mathit{index\_term}_n)\,[y(\mathtt{int}\, z)\,/y_1, .., y(\mathtt{int}\, z)\,/y_n]} \quad \textsc{Val\_Obj\_Arr}$$

$$\frac{}{\Gamma; \Lambda; \Xi \vdash \mathtt{Unit} : y, \mathtt{unit}, y = ()} \quad \textsc{Val\_Unit}$$

$$\frac{}{\Gamma; \Lambda; \Xi \vdash \mathtt{True} : y, \mathtt{bool}, y = \mathtt{true}} \quad \textsc{Val\_True}$$

$$\frac{}{\Gamma; \Lambda; \Xi \vdash \mathtt{False} : y, \mathtt{bool}, y = \mathtt{false}} \quad \textsc{Val\_False}$$

$$\frac{\Gamma; \Lambda; \Xi \vdash \mathit{value}_1 : y_1, \beta, \mathit{index\_term}_1 \quad .. \quad \Gamma; \Lambda; \Xi \vdash \mathit{value}_n : y_n, \beta, \mathit{index\_term}_n}{\Gamma; \Lambda; \Xi \vdash \beta[\mathit{value}_1, .., \mathit{value}_n] : y, [\beta], (\bigwedge(\mathit{index\_term}_1, .., \mathit{index\_term}_n))[y^{(k)}\,/y_1, .., y^{(k)}\,/y_n]} \quad \textsc{Val\_List}$$

$$\frac{\Gamma; \Lambda; \Xi \vdash \mathit{value}_1 : y_1, \beta_1, \mathit{index\_term}_1 \quad .. \quad \Gamma; \Lambda; \Xi \vdash \mathit{value}_n : y_n, \beta_n, \mathit{index\_term}_n}{\Gamma; \Lambda; \Xi \vdash (\mathit{value}_1, .., \mathit{value}_n) : y, (\beta_1, .., \beta_n), \bigwedge(\mathit{index\_term}_1, .., \mathit{index\_term}_n)\,[y^{(k)}\,/y_1, .., y^{(k)}\,/y_n]} \quad \textsc{Val\_Tuple}$$

$\boxed{\Gamma; \Lambda; \Xi \vdash \mathit{pexpr\_aux} : \mathit{ret}}$

$$\frac{x : \beta \in \Gamma}{\Gamma; \Lambda; \Xi \vdash x : \Sigma\, y : \beta.\mathtt{I}} \quad \textsc{PExpr\_Var}$$

$$\frac{\Gamma; \Lambda; \Xi \vdash \mathit{value} : y, \beta, \mathit{index\_term}}{\Gamma; \Lambda; \Xi \vdash \mathit{value} : \Sigma\, y : \beta.\mathit{index\_term} \wedge \mathtt{I}} \quad \textsc{PExpr\_Val}$$

$$\frac{x : \mathtt{bool} \in \Gamma}{\Gamma; \Lambda; \Xi \vdash \mathtt{not}\,(x) : \Sigma\, y : \mathtt{bool}.y = (\neg\, x\,) \wedge \mathtt{I}} \quad \textsc{PExpr\_Not}$$

```
Definition rules:        13 good    0 bad
Definition rule clauses: 19 good    0 bad
```