

<i>n, i, j, k</i>	Index variables for meta-lists
<i>num</i>	Numeric literals
<i>hex</i>	Bit vector literal, specified by C-style hex number
<i>bin</i>	Bit vector literal, specified by C-style binary number
<i>string</i>	String literals
<i>regexp</i>	Regular expressions, as a string literal
<i>x, y, z</i>	Variables
<i>ix</i>	Variables

l	$::=$ 	Source locations
$x^l, y^l, z^l, name$	$::=$ $x\ l$ $(ix)l$	Location-annotated names Remove infix status
ix^l	$::=$ $ix\ l$ $'x' l$	Location-annotated infix names Add infix status
α	$::=$ $'x$	Type variables
α^l	$::=$ $\alpha\ l$	Location-annotated type variables
N	$::=$ $''x$	numeric variables
N^l	$::=$ $N\ l$	Location-annotated numeric variables
id	$::=$ $x_1^l \dots x_n^l . x^l\ l$	Long identifiers
tnv	$::=$ α N	Union of type variables and Nexp type variables, without location
$tnvar^l$	$::=$ α^l N^l	Union of type variables and Nexp type variables, with location
$tnvs$	$::=$ $tnv_1 .. tnv_n$	Type variable lists
$tnvars^l$	$::=$ $tnvar_1^l .. tnvar_n^l$	Type variable lists
$Nexp_aux$	$::=$ N num $Nexp_1 * Nexp_2$ $Nexp_1 + Nexp_2$ $(Nexp)$	Numerical expressions for specifying vector lengths and indexes

$Nexp$	$::=$ $ \quad Nexp_aux \ l$	Location-annotated vector lengths
$Nexp_constraint$	$::=$ $ \quad Nexp$ $ \quad \geq Nexp$	Whether a vector is bounded or fixed size
typ_aux	$::=$ $ \quad -$ $ \quad \alpha^l$ $ \quad typ_1 \rightarrow typ_2$ $ \quad typ_1 * \dots * typ_n$ $ \quad Nexp$ $ \quad id \ typ_1 .. typ_n$ $ \quad (typ)$	Types Unspecified type Type variables Function types Tuple types As a typ to permit applications over Nexps, otherwise no Type applications
typ	$::=$ $ \quad typ_aux \ l$	Location-annotated types
lit_aux	$::=$ $ \quad \mathbf{true}$ $ \quad \mathbf{false}$ $ \quad num$ $ \quad hex$ $ \quad bin$ $ \quad string$ $ \quad ()$ $ \quad \mathbf{bitzero}$ $ \quad \mathbf{bitone}$	Literal constants hex and bin are constant bit vectors, entered as C-style L bitzero and bitone are constant bits, if commonly used w
lit	$::=$ $ \quad lit_aux \ l$	Location-annotated literal constants
$;\text{?}$	$::=$ $ $ $ \quad ;$	Optional semi-colons
pat_aux	$::=$ $ \quad -$ $ \quad (pat \ \mathbf{as} \ x^l)$ $ \quad (pat : typ)$ $ \quad id \ pat_1 .. pat_n$ $ \quad \langle fpat_1; \dots; fpat_n; ? \rangle$ $ \quad [pat_1; \dots; pat_n; ?]$ $ \quad [pat_1 .. pat_n]$ $ \quad (pat_1, \dots, pat_n)$ $ \quad [pat_1; \dots; pat_n; ?]$	Patterns Wildcards Named patterns Typed patterns Single variable and constructor patterns Record patterns Vector patterns Concatenated vector patterns Tuple patterns List patterns

	(pat) $pat_1 :: pat_2$ $x^l + num$ lit	Cons patterns constant addition patterns Literal constant patterns
pat	$::=$ $pat_aux\ l$	Location-annotated patterns
$fpat$	$::=$ $id = pat\ l$	Field patterns
$?$	$::=$ $ $ $ $	Optional bars
exp_aux	$::=$ id N $\mathbf{fun}\ psexp$ $\mathbf{function}\ ^? pexp_1 \dots pexp_n\ \mathbf{end}$ $exp_1\ exp_2$ $exp_1\ ix^l\ exp_2$ $\langle fexp_s \rangle$ $\langle exp\ \mathbf{with}\ fexp_s \rangle$ $exp.id$ $[exp_1; \dots; exp_n; ?]$ $exp.(Nexp)$ $exp.(Nexp_1..Nexp_2)$ $\mathbf{match}\ exp\ \mathbf{with}\ ^? pexp_1 \dots pexp_n\ l\ \mathbf{end}$ $(exp : typ)$ $\mathbf{let}\ letbind\ \mathbf{in}\ exp$ (exp_1, \dots, exp_n) $[exp_1; \dots; exp_n; ?]$ (exp) $\mathbf{begin}\ exp\ \mathbf{end}$ $\mathbf{if}\ exp_1\ \mathbf{then}\ exp_2\ \mathbf{else}\ exp_3$ $exp_1 :: exp_2$ lit $\{exp_1 exp_2\}$ $\{exp_1 \mathbf{forall}\ qbind_1 \dots qbind_n exp_2\}$ $\{exp_1; \dots; exp_n; ?\}$ $q\ qbind_1 \dots qbind_n.exp$ $[exp_1 \mathbf{forall}\ qbind_1 \dots qbind_n exp_2]$	Expressions Identifiers Nexp var, has type num Curried functions Functions with pattern matching Function applications Infix applications Records Functional update for records Field projection for records Vector instantiation Vector access Subvector extraction Pattern matching expressions Type-annotated expressions Let expressions Tuples Lists Alternate syntax for (exp) Conditionals Cons expressions Literal constants Set comprehensions Set comprehensions with explicit binding Sets Logical quantifications List comprehensions (all binders must be qualified)
exp	$::=$ $exp_aux\ l$	Location-annotated expressions

q	$::=$ \mid forall \mid exists	Quantifiers
$qbind$	$::=$ $\mid x^l$ $\mid (pat \textbf{IN} exp)$ $\mid (pat \textbf{MEM} exp)$	Bindings for quantifiers Restricted quantifications over sets Restricted quantifications over lists
$fexp$	$::=$ $\mid id = exp \ l$	Field-expressions
$fexps$	$::=$ $\mid fexp_1; \dots; fexp_n; ? \ l$	Field-expression lists
$pexp$	$::=$ $\mid pat \rightarrow exp \ l$	Pattern matches
$psexp$	$::=$ $\mid pat_1 \dots pat_n \rightarrow exp \ l$	Multi-pattern matches
$tannot^?$	$::=$ \mid $\mid : typ$	Optional type annotations
$funcl_aux$	$::=$ $\mid x^l \ pat_1 \dots pat_n \ tannot^? = exp$	Function clauses
$letbind_aux$	$::=$ $\mid pat \ tannot^? = exp$ $\mid funcl_aux$	Let bindings Value bindings Function bindings
$letbind$	$::=$ $\mid letbind_aux \ l$	Location-annotated let bindings
$funcl$	$::=$ $\mid funcl_aux \ l$	Location-annotated function clauses
$id^?$	$::=$ \mid $\mid x^l :$	Optional name for inductively defined relations
$rule_aux$	$::=$ $\mid id^? \textbf{forall} \ x_1^l \dots x_n^l. exp \implies x^l \ exp_1 \dots exp_i$	Inductively defined relation clauses
$rule$	$::=$ $\mid rule_aux \ l$	Location-annotated inductively defined relations

<i>typs</i>	$::=$ $ \quad typ_1 * \dots * typ_n$	Type lists
<i>ctor_def</i>	$::=$ $ \quad x^l \textbf{of } typs$ $ \quad x^l$	Datatype definition clauses S Constant constructors
<i>texp</i>	$::=$ $ \quad typ$ $ \quad \langle x_1^l : typ_1; \dots; x_n^l : typ_n; ? \rangle$ $ \quad ^? ctor_def_1 \dots ctor_def_n$	Type definition bodies Type abbreviations Record types Variant types
<i>name?</i>	$::=$ $ $ $ \quad [name = regexp]$	Optional name specification for variables of defined type
<i>td</i>	$::=$ $ \quad x^l tnvars^l name^? = texp$ $ \quad x^l tnvars^l name^?$	Type definitions Definitions of opaque types
<i>c</i>	$::=$ $ \quad id tnvar^l$	Typeclass constraints
<i>cs</i>	$::=$ $ $ $ \quad c_1, \dots, c_i \Rightarrow$	Typeclass constraint lists Must have > 0 constraints
<i>c_pre</i>	$::=$ $ $ $ \quad \textbf{forall } tnvar_1^l \dots tnvar_n^l . cs$	Type and instance scheme prefixes Must have > 0 type variables
<i>typschm</i>	$::=$ $ \quad c_pre \, typ$	Type schemes
<i>instschm</i>	$::=$ $ \quad c_pre(id \, typ)$	Instance schemes
<i>target</i>	$::=$ $ \quad \textbf{hol}$ $ \quad \textbf{isabelle}$ $ \quad \textbf{ocaml}$ $ \quad \textbf{coq}$ $ \quad \textbf{tex}$ $ \quad \textbf{html}$	Backend target names
τ	$::=$ $ \quad \{target_1; \dots; target_n\}$	Backend target name lists

$\tau^?$	$::=$ $ $ $ \quad \tau$	Optional targets
<i>lemma_typ</i>	$::=$ $ \quad \mathbf{assert}$ $ \quad \mathbf{lemma}$ $ \quad \mathbf{theorem}$	Types of Lemmata
<i>lemma_decl</i>	$::=$ $ \quad \mathit{lemma_typ} \ \tau^? \ x^l : (\mathit{exp})$ $ \quad \mathit{lemma_typ} \ \tau^? (\mathit{exp})$	Lemmata and Tests
<i>val_def</i>	$::=$ $ \quad \mathbf{let} \ \tau^? \ \mathit{letbind}$ $ \quad \mathbf{let \ rec} \ \tau^? \ \mathit{funcl}_1 \mathbf{and} \dots \mathbf{and} \ \mathit{funcl}_n$ $ \quad \mathbf{let \ inline} \ \tau^? \ \mathit{letbind}$	Value definitions Non-recursive value definitions Recursive function definitions Function definitions to be inlined
<i>val_spec</i>	$::=$ $ \quad \mathbf{val} \ x^l : \mathit{typschm}$	Value type specifications
<i>def_aux</i>	$::=$ $ \quad \mathbf{type} \ \mathit{td}_1 \mathbf{and} \dots \mathbf{and} \ \mathit{td}_n$ $ \quad \mathit{val_def}$ $ \quad \mathit{lemma_decl}$ $ \quad \mathbf{rename} \ \tau^? \ \mathit{id} = x^l$ $ \quad \mathbf{module} \ x^l = \mathbf{struct} \ \mathit{defs} \mathbf{end}$ $ \quad \mathbf{module} \ x^l = \mathit{id}$ $ \quad \mathbf{open} \ \mathit{id}$ $ \quad \mathbf{indreln} \ \tau^? \ \mathit{rule}_1 \mathbf{and} \dots \mathbf{and} \ \mathit{rule}_n$ $ \quad \mathit{val_spec}$ $ \quad \mathbf{class} \ (x^l \ \mathit{tnvar}^l) \mathbf{val} \ x_1^l : \mathit{typ}_1 \ l_1 \dots \mathbf{val} \ x_n^l : \mathit{typ}_n \ l_n \mathbf{end}$ $ \quad \mathbf{instance} \ \mathit{instschm} \ \mathit{val_def}_1 \ l_1 \dots \mathit{val_def}_n \ l_n \mathbf{end}$	Top-level definitions Type definitions Value definitions Lemmata Rename constant or type Module definitions Module renamings Opening modules Inductively defined relations Top-level type constraints Typeclass definitions Typeclass instantiations
<i>def</i>	$::=$ $ \quad \mathit{def_aux} \ l$	Location-annotated definitions
$;;^?$	$::=$ $ $ $ \quad ;;$	Optional double-semi-colon
<i>defs</i>	$::=$ $ \quad \mathit{def}_1 ; ;_1^? \dots \mathit{def}_n ; ;_n^?$	Definition sequences
<i>p</i>	$::=$ $ \quad x_1 \dots x_n . x$ $ \quad \mathbf{_list}$	Unique paths

		--bool	
		--num	
		--set	
		--string	
		--unit	
		--bit	
		--vector	
σ	::=		Type variable substitutions
		$\{tnv_1 \mapsto t_1 \dots tnv_n \mapsto t_n\}$	
t, u	::=		Internal types
		α	
		$t_1 \rightarrow t_2$	
		$t_1 * \dots * t_n$	
		$p \ t_args$	
		ne	
		$\sigma(t)$	M Multiple substitutions
		$\sigma(tnv)$	M Single variable substitution
		curry (t_multi, t)	M Curried, multiple argument functions
ne	::=		internal numeric expressions
		N	
		num	
		$num * ne$	
		$ne_1 + ne_2$	
		$(-ne)$	
		normalize (ne)	M
		$ne_1 + \dots + ne_n$	M
		bitlength (bin)	M
		bitlength (hex)	M
		length ($pat_1 \dots pat_n$)	M
		length ($exp_1 \dots exp_n$)	M
t_args	::=		Lists of types
		$t_1 \dots t_n$	
		$\sigma(t_args)$	M Multiple substitutions
t_multi	::=		Lists of types
		$(t_1 * \dots * t_n)$	
		$\sigma(t_multi)$	M Multiple substitutions
nec	::=		Numeric expression constraints
		$ne \langle nec$	
		$ne = nec$	
		$ne \leq nec$	
		ne	

$names$	$::=$ $\{x_1, \dots, x_n\}$	Sets of names
\mathcal{C}	$::=$ $(p_1 \text{ } \textit{tnv}_1) \dots (p_n \text{ } \textit{tnv}_n)$	Typeclass constraint lists
env_tag	$::=$ method val let	Tags for the (non-constructor) value descriptions Bound to a method Specified with val Defined with let or indreln
v_desc	$::=$ $\langle \textbf{forall } \textit{tnvs}.t_multi \rightarrow p, (x \textbf{ of } names) \rangle$ $\langle \textbf{forall } \textit{tnvs}.\mathcal{C} \Rightarrow t, env_tag \rangle$	Value descriptions Constructors Values
f_desc	$::=$ $\langle \textbf{forall } \textit{tnvs}.p \rightarrow t, (x \textbf{ of } names) \rangle$	Fields
$\Sigma^{\mathcal{C}}$	$::=$ $\{(p_1 \text{ } t_1), \dots, (p_n \text{ } t_n)\}$ $\Sigma^{\mathcal{C}}_1 \cup \dots \cup \Sigma^{\mathcal{C}}_n$	Typeclass constraints M
$\Sigma^{\mathcal{N}}$	$::=$ $\{nec_1, \dots, nec_n\}$ $\Sigma^{\mathcal{N}}_1 \cup \dots \cup \Sigma^{\mathcal{N}}_n$	Nexp constraint lists M
E	$::=$ $\langle E^M, E^P, E^F, E^X \rangle$ $E_1 \uplus E_2$ ϵ	Environments M M
E^X	$::=$ $\{x_1 \mapsto v_desc_1, \dots, x_n \mapsto v_desc_n\}$ $E_1^X \uplus \dots \uplus E_n^X$	Value environments M
E^F	$::=$ $\{x_1 \mapsto f_desc_1, \dots, x_n \mapsto f_desc_n\}$ $E_1^F \uplus \dots \uplus E_n^F$	Field environments M
E^M	$::=$ $\{x_1 \mapsto E_1, \dots, x_n \mapsto E_n\}$	Module environments
E^P	$::=$ $\{x_1 \mapsto p_1, \dots, x_n \mapsto p_n\}$ $E_1^P \uplus \dots \uplus E_n^P$	Path environments M
E^L	$::=$	Lexical bindings

	$\begin{array}{ l} \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\} \\ E_1^L \uplus \dots \uplus E_n^L \end{array}$	M	
<i>tc_abbrev</i>	$\begin{array}{ l} ::= \\ .t \end{array}$		Type abbreviations
<i>tc_def</i>	$\begin{array}{ l} ::= \\ tnvs\ tc_abbrev \end{array}$		Type and class constructor definitions Type constructors
Δ	$\begin{array}{ l} ::= \\ \{p_1 \mapsto tc_def_1, \dots, p_n \mapsto tc_def_n\} \\ \Delta_1 \uplus \Delta_2 \end{array}$	M	Type constructor definitions
δ	$\begin{array}{ l} ::= \\ \{p_1 \mapsto xs_1, \dots, p_n \mapsto xs_n\} \\ \delta_1 \uplus \delta_2 \end{array}$	M	Typeclass definitions
<i>inst</i>	$\begin{array}{ l} ::= \\ \mathcal{C} \Rightarrow (p\ t) \end{array}$		A typeclass instance, t must not contain nested type
<i>I</i>	$\begin{array}{ l} ::= \\ \{inst_1, \dots, inst_n\} \\ I_1 \cup I_2 \end{array}$	M	Global instances
<i>D</i>	$\begin{array}{ l} ::= \\ \langle \Delta, \delta, I \rangle \\ D_1 \uplus D_2 \\ \epsilon \end{array}$	M M	Global type definition store
<i>xs</i>	$\begin{array}{ l} ::= \\ x_1 \dots x_n \end{array}$		
<i>terminals</i>	$\begin{array}{ l} ::= \\ \geq \\ \rightarrow \\ \Rightarrow \\ \langle \\ \rangle \\ \cap \\ \cup \\ \uplus \\ \notin \\ \subset \\ \neq \\ \emptyset \\ \langle \end{array}$	$\begin{array}{ l} >= \\ -> \\ ==> \\ < \\ > \end{array}$	

		\rangle	
		\vdash	
		$,$	
		\mapsto	
		\triangleright	
		\rightsquigarrow	
		\Rightarrow	
		$-$	
		ϵ	
<i>formula</i>	$::=$		
		<i>judgement</i>	
		$formula_1 \dots formula_n$	
		$E^M(x) \triangleright E$	Module lookup
		$E^P(x) \triangleright p$	Path lookup
		$E^F(x) \triangleright f_desc$	Field lookup
		$E^X(x) \triangleright v_desc$	Value lookup
		$E^L(x) \triangleright t$	Lexical binding lookup
		$\Delta(p) \triangleright tc_def$	Type constructor lookup
		$\delta(p) \triangleright xs$	Type constructor lookup
		$\mathbf{dom}(E_1^M) \cap \mathbf{dom}(E_2^M) = \emptyset$	
		$\mathbf{dom}(E_1^X) \cap \mathbf{dom}(E_2^X) = \emptyset$	
		$\mathbf{dom}(E_1^F) \cap \mathbf{dom}(E_2^F) = \emptyset$	
		$\mathbf{dom}(E_1^P) \cap \mathbf{dom}(E_2^P) = \emptyset$	
		$\mathbf{disjoint\ doms}(E_1^L, \dots, E_n^L)$	Pairwise disjoint domains
		$\mathbf{disjoint\ doms}(E_1^X, \dots, E_n^X)$	Pairwise disjoint domains
		$\mathbf{compatible\ overlap}(x_1 \mapsto t_1, \dots, x_n \mapsto t_n)$	$(x_i = x_j) \implies (t_i = t_j)$
		$\mathbf{duplicates}(tnvs) = \emptyset$	
		$\mathbf{duplicates}(x_1, \dots, x_n) = \emptyset$	
		$x \notin \mathbf{dom}(E^L)$	
		$x \notin \mathbf{dom}(E^X)$	
		$x \notin \mathbf{dom}(E^F)$	
		$p \notin \mathbf{dom}(\delta)$	
		$p \notin \mathbf{dom}(\Delta)$	
		$\mathbf{FV}(t) \subset tnvs$	Free type variables
		$\mathbf{FV}(t_multi) \subset tnvs$	Free type variables
		$\mathbf{FV}(\mathcal{C}) \subset tnvs$	Free type variables
		<i>inst</i> IN <i>I</i>	
		$(p\ t) \notin I$	
		$E_1^L = E_2^L$	
		$E_1^X = E_2^X$	
		$E_1^F = E_2^F$	
		$E_1 = E_2$	
		$\Delta_1 = \Delta_2$	
		$\delta_1 = \delta_2$	
		$I_1 = I_2$	

	$ \begin{array}{ l} names_1 = names_2 \\ t_1 = t_2 \\ \sigma_1 = \sigma_2 \\ p_1 = p_2 \\ xs_1 = xs_2 \\ tnvs_1 = tnvs_2 \end{array} $	
<i>convert_tnvars</i>	$ \begin{array}{ l} tnvars^l \rightsquigarrow tnvs \\ tnvar^l \rightsquigarrow tn timer \end{array} $	
<i>look_m</i>	$ \begin{array}{ l} E_1(x_1^l \dots x_n^l) \triangleright E_2 \end{array} $	Name path lookup
<i>look_m_id</i>	$ \begin{array}{ l} E_1(id) \triangleright E_2 \end{array} $	Module identifier lookup
<i>look_tc</i>	$ \begin{array}{ l} E(id) \triangleright p \end{array} $	Path identifier lookup
<i>check_t</i>	$ \begin{array}{ l} \Delta \vdash t \textbf{ok} \\ \Delta, tn timer \vdash t \textbf{ok} \end{array} $	Well-formed types Well-formed type/Nexps m
<i>teq</i>	$ \begin{array}{ l} \Delta \vdash t_1 = t_2 \end{array} $	Type equality
<i>convert_typ</i>	$ \begin{array}{ l} \Delta, E \vdash typ \rightsquigarrow t \\ \vdash Nexp \rightsquigarrow ne \end{array} $	Convert source types to im Convert and normalize num
<i>convert_typs</i>	$ \begin{array}{ l} \Delta, E \vdash typs \rightsquigarrow t_multi \end{array} $	
<i>check_lit</i>	$ \begin{array}{ l} \vdash lit : t \end{array} $	Typing literal constants
<i>inst_field</i>	$ \begin{array}{ l} \Delta, E \vdash \textbf{field } id : p \ t_args \rightarrow t \triangleright (x \textbf{ of } names) \end{array} $	Field typing (also returns c
<i>inst_ctor</i>	$ \begin{array}{ l} \Delta, E \vdash \textbf{ctor } id : t_multi \rightarrow p \ t_args \triangleright (x \textbf{ of } names) \end{array} $	Data constructor typing (a
<i>inst_val</i>	$ \begin{array}{ l} \Delta, E \vdash \textbf{val } id : t \triangleright \Sigma^C \end{array} $	Typing top-level bindings,
<i>not_ctor</i>	$ \begin{array}{ l} \vdash id : t \triangleright \Sigma^C \end{array} $	

		$E, E^L \vdash x$ not ctor	v is not bound to a data constructor
<i>not_shadowed</i>	::=	$E^L \vdash id$ not shadowed	id is not lexically shadowed
<i>check_pat</i>	::=	$\Delta, E, E_1^L \vdash pat : t \triangleright E_2^L$ $\Delta, E, E_1^L \vdash pat_aux : t \triangleright E_2^L$	Typing patterns, building the environment Typing patterns, building the environment
<i>id_field</i>	::=	$E \vdash id$ field	Check that the identifier is a field
<i>id_value</i>	::=	$E \vdash id$ value	Check that the identifier is a value
<i>check_exp</i>	::=	$\Delta, E, E^L \vdash exp : t \triangleright \Sigma^C, \Sigma^N$ $\Delta, E, E^L \vdash exp_aux : t \triangleright \Sigma^C, \Sigma^N$ $\Delta, E, E_1^L \vdash qbind_1 .. qbind_n \triangleright E_2^L, \Sigma^C$ $\Delta, E, E_1^L \vdash \mathbf{list} \ qbind_1 .. qbind_n \triangleright E_2^L, \Sigma^C$ $\Delta, E, E^L \vdash func1 \triangleright \{x \mapsto t\}, \Sigma^C, \Sigma^N$ $\Delta, E, E_1^L \vdash letbind \triangleright E_2^L, \Sigma^C, \Sigma^N$	Typing expressions, collecting constraints Typing expressions, collecting constraints Build the environment for qualified identifiers Build the environment for qualified identifiers Build the environment for a function Build the environment for a let binding
<i>check_rule</i>	::=	$\Delta, E, E^L \vdash rule \triangleright \{x \mapsto t\}, \Sigma^C, \Sigma^N$	Build the environment for an environment
<i>check_texp_tc</i>	::=	$xs, \Delta_1, E \vdash \mathbf{tc} \ td \triangleright \Delta_2, E^P$	Extract the type constructor from a type expression
<i>check_texprs_tc</i>	::=	$xs, \Delta_1, E \vdash \mathbf{tc} \ td_1 .. td_i \triangleright \Delta_2, E^P$	Extract the type constructor from a type expression
<i>check_texp</i>	::=	$\Delta, E \vdash tnvs \ p = texp \triangleright \langle E^F, E^X \rangle$	Check a type definition, with fresh variables
<i>check_texprs</i>	::=	$xs, \Delta, E \vdash td_1 .. td_n \triangleright \langle E^F, E^X \rangle$	Check a type definition, with fresh variables
<i>convert_class</i>	::=	$\delta, E \vdash id \rightsquigarrow p$	Lookup a type class
<i>solve_class_constraint</i>	::=	$I \vdash (p \ t) \mathbf{IN} \ C$	Solve class constraint
<i>solve_class_constraints</i>	::=	$I \vdash \Sigma^C \triangleright C$	Solve class constraints

<i>check_val_def</i>	$::=$ $\mid \Delta, I, E \vdash \textit{val_def} \triangleright E^x$	Check a value definition
<i>check_t_instance</i>	$::=$ $\mid \Delta, (\alpha_1, \dots, \alpha_n) \vdash t \textbf{instance}$	Check that t be a typeclass instance
<i>check_defs</i>	$::=$ $\mid \overline{z_j^j}, D_1, E_1 \vdash \textit{def} \triangleright D_2, E_2$ $\mid \overline{z_j^j}, D_1, E_1 \vdash \textit{defs} \triangleright D_2, E_2$	Check a definition Check definitions, given module path, definitions
<i>judgement</i>	$::=$ $\mid \textit{convert_tnvars}$ $\mid \textit{look_m}$ $\mid \textit{look_m_id}$ $\mid \textit{look_tc}$ $\mid \textit{check_t}$ $\mid \textit{teq}$ $\mid \textit{convert_typ}$ $\mid \textit{convert_typs}$ $\mid \textit{check_lit}$ $\mid \textit{inst_field}$ $\mid \textit{inst_ctor}$ $\mid \textit{inst_val}$ $\mid \textit{not_ctor}$ $\mid \textit{not_shadowed}$ $\mid \textit{check_pat}$ $\mid \textit{id_field}$ $\mid \textit{id_value}$ $\mid \textit{check_exp}$ $\mid \textit{check_rule}$ $\mid \textit{check_terp_tc}$ $\mid \textit{check_terps_tc}$ $\mid \textit{check_terp}$ $\mid \textit{check_terps}$ $\mid \textit{convert_class}$ $\mid \textit{solve_class_constraint}$ $\mid \textit{solve_class_constraints}$ $\mid \textit{check_val_def}$ $\mid \textit{check_t_instance}$ $\mid \textit{check_defs}$	
<i>user_syntax</i>	$::=$ $\mid n$ $\mid \textit{num}$ $\mid \textit{hex}$ $\mid \textit{bin}$ $\mid \textit{string}$	

	<i>regeexp</i>
	<i>x</i>
	<i>ix</i>
	<i>l</i>
	<i>x^l</i>
	<i>ix^l</i>
	α
	α^l
	<i>N</i>
	<i>N^l</i>
	<i>id</i>
	<i>tnv</i>
	<i>tnvar^l</i>
	<i>tnvs</i>
	<i>tnvars^l</i>
	<i>Nexp_aux</i>
	<i>Nexp</i>
	<i>Nexp_constraint</i>
	<i>typ_aux</i>
	<i>typ</i>
	<i>lit_aux</i>
	<i>lit</i>
	<i>;</i> [?]
	<i>pat_aux</i>
	<i>pat</i>
	<i>fpat</i>
	<i> </i> [?]
	<i>exp_aux</i>
	<i>exp</i>
	<i>q</i>
	<i>qbind</i>
	<i>fexp</i>
	<i>fexps</i>
	<i>pexp</i>
	<i>psexp</i>
	<i>tannot</i> [?]
	<i>funcl_aux</i>
	<i>letbind_aux</i>
	<i>letbind</i>
	<i>funcl</i>
	<i>id</i> [?]
	<i>rule_aux</i>
	<i>rule</i>
	<i>typs</i>
	<i>ctor_def</i>
	<i>terp</i>

	<i>name?</i>
	<i>td</i>
	<i>c</i>
	<i>cs</i>
	<i>c_pre</i>
	<i>typschm</i>
	<i>instschm</i>
	<i>target</i>
	τ
	$\tau?$
	<i>lemma_typ</i>
	<i>lemma_decl</i>
	<i>val_def</i>
	<i>val_spec</i>
	<i>def_aux</i>
	<i>def</i>
	$::?$
	<i>defs</i>
	<i>p</i>
	σ
	<i>t</i>
	<i>ne</i>
	<i>t_args</i>
	<i>t_multi</i>
	<i>nec</i>
	<i>names</i>
	\mathcal{C}
	<i>env_tag</i>
	<i>v_desc</i>
	<i>f_desc</i>
	$\Sigma^{\mathcal{C}}$
	$\Sigma^{\mathcal{N}}$
	E
	$E^{\mathbf{x}}$
	$E^{\mathbf{F}}$
	$E^{\mathbf{M}}$
	$E^{\mathbf{P}}$
	$E^{\mathbf{L}}$
	<i>tc_abbrev</i>
	<i>tc_def</i>
	Δ
	δ
	<i>inst</i>
	<i>I</i>
	<i>D</i>
	<i>xs</i>

| *terminals*
| *formula*

$tnvars^l \rightsquigarrow tnvs$

$$\frac{tnvar_1^l \rightsquigarrow tn v_1 \quad \dots \quad tnvar_n^l \rightsquigarrow tn v_n}{tnvar_1^l \dots tnvar_n^l \rightsquigarrow tn v_1 \dots tn v_n} \quad \text{CONVERT_TNVARS_NONE}$$

$tnvar^l \rightsquigarrow tn v$

$$\frac{}{\alpha \, l \rightsquigarrow \alpha} \quad \text{CONVERT_TNVAR_A}$$

$$\frac{}{N \, l \rightsquigarrow N} \quad \text{CONVERT_TNVAR_N}$$

$E_1(x_1^l \dots x_n^l) \triangleright E_2$

Name path lookup

$$\frac{}{E() \triangleright E} \quad \text{LOOK_M_NONE}$$

$$\frac{\begin{array}{c} E^M(x) \triangleright E_1 \\ E_1(\overline{y_i^l}^i) \triangleright E_2 \end{array}}{\langle E^M, E^P, E^F, E^X \rangle (x \, l \, \overline{y_i^l}^i) \triangleright E_2} \quad \text{LOOK_M_SOME}$$

$E_1(id) \triangleright E_2$

Module identifier lookup

$$\frac{E_1(\overline{y_i^l}^i \, x \, l_1) \triangleright E_2}{E_1(\overline{y_i^l}^i \, x \, l_1 \, l_2) \triangleright E_2} \quad \text{LOOK_M_ID_ALL}$$

$E(id) \triangleright p$

Path identifier lookup

$$\frac{\begin{array}{c} E(\overline{y_i^l}^i) \triangleright \langle E^M, E^P, E^F, E^X \rangle \\ E^P(x) \triangleright p \end{array}}{E(\overline{y_i^l}^i \, x \, l_1 \, l_2) \triangleright p} \quad \text{LOOK_TC_ALL}$$

$\Delta \vdash t \mathbf{ok}$

Well-formed types

$$\frac{}{\Delta \vdash \alpha \mathbf{ok}} \quad \text{CHECK_T_VAR}$$

$$\frac{\begin{array}{c} \Delta \vdash t_1 \mathbf{ok} \\ \Delta \vdash t_2 \mathbf{ok} \end{array}}{\Delta \vdash t_1 \rightarrow t_2 \mathbf{ok}} \quad \text{CHECK_T_FN}$$

$$\frac{\Delta \vdash t_1 \mathbf{ok} \quad \dots \quad \Delta \vdash t_n \mathbf{ok}}{\Delta \vdash t_1 * \dots * t_n \mathbf{ok}} \quad \text{CHECK_T_TUP}$$

$$\frac{\begin{array}{c} \Delta(p) \triangleright tn v_1 \dots tn v_n \, tc_abbrev \\ \Delta, tn v_1 \vdash t_1 \mathbf{ok} \quad \dots \quad \Delta, tn v_n \vdash t_n \mathbf{ok} \end{array}}{\Delta \vdash p \, t_1 \dots t_n \mathbf{ok}} \quad \text{CHECK_T_APP}$$

$\Delta, tn v \vdash t \mathbf{ok}$

Well-formed type/Nexps matching the application type variable

$$\frac{\Delta \vdash t \mathbf{ok}}{\Delta, \alpha \vdash t \mathbf{ok}} \quad \text{CHECK_TLEN_T}$$

$$\overline{\Delta, N \vdash ne \mathbf{ok}} \quad \text{CHECK_TLEN_LEN}$$

$\boxed{\Delta \vdash t_1 = t_2}$ Type equality

$$\begin{array}{c} \frac{\Delta \vdash t \mathbf{ok}}{\Delta \vdash t = t} \quad \text{TEQ_REFL} \\[10pt] \frac{\Delta \vdash t_2 = t_1}{\Delta \vdash t_1 = t_2} \quad \text{TEQ_SYM} \\[10pt] \frac{\Delta \vdash t_1 = t_2 \quad \Delta \vdash t_2 = t_3}{\Delta \vdash t_1 = t_3} \quad \text{TEQ_TRANS} \\[10pt] \frac{\Delta \vdash t_1 = t_3 \quad \Delta \vdash t_2 = t_4}{\Delta \vdash t_1 \rightarrow t_2 = t_3 \rightarrow t_4} \quad \text{TEQ_ARROW} \\[10pt] \frac{\Delta \vdash t_1 = u_1 \quad \dots \quad \Delta \vdash t_n = u_n}{\Delta \vdash t_1 * \dots * t_n = u_1 * \dots * u_n} \quad \text{TEQ_TUP} \\[10pt] \frac{\Delta(p) \triangleright \alpha_1 .. \alpha_n \quad \Delta \vdash t_1 = u_1 \quad \dots \quad \Delta \vdash t_n = u_n}{\Delta \vdash p \, t_1 .. t_n = p \, u_1 .. u_n} \quad \text{TEQ_APP} \\[10pt] \frac{\Delta(p) \triangleright \alpha_1 .. \alpha_n . u}{\Delta \vdash p \, t_1 .. t_n = \{\alpha_1 \mapsto t_1 .. \alpha_n \mapsto t_n\}(u)} \quad \text{TEQ_EXPAND} \\[10pt] \frac{ne = \mathbf{normalize}(ne')}{\Delta \vdash ne = ne'} \quad \text{TEQ_NEXP} \end{array}$$

$\boxed{\Delta, E \vdash typ \rightsquigarrow t}$ Convert source types to internal types

$$\begin{array}{c} \overline{\Delta, E \vdash \alpha \, l' \, l \rightsquigarrow \alpha} \quad \text{CONVERT_TYP_VAR} \\[10pt] \frac{\Delta, E \vdash typ_1 \rightsquigarrow t_1 \quad \Delta, E \vdash typ_2 \rightsquigarrow t_2}{\Delta, E \vdash typ_1 \rightarrow typ_2 \, l \rightsquigarrow t_1 \rightarrow t_2} \quad \text{CONVERT_TYP_FN} \\[10pt] \frac{\Delta, E \vdash typ_1 \rightsquigarrow t_1 \quad \dots \quad \Delta, E \vdash typ_n \rightsquigarrow t_n}{\Delta, E \vdash typ_1 * \dots * typ_n \, l \rightsquigarrow t_1 * \dots * t_n} \quad \text{CONVERT_TYP_TUP} \\[10pt] \frac{\Delta, E \vdash typ_1 \rightsquigarrow t_1 \quad \dots \quad \Delta, E \vdash typ_n \rightsquigarrow t_n \quad E(id) \triangleright p \quad \Delta(p) \triangleright \alpha_1 .. \alpha_n \, tc_abbrev}{\Delta, E \vdash id \, typ_1 .. typ_n \, l \rightsquigarrow p \, t_1 .. t_n} \quad \text{CONVERT_TYP_APP} \\[10pt] \frac{\vdash Nexp \rightsquigarrow ne}{\Delta, E \vdash Nexp \rightsquigarrow ne} \quad \text{CONVERT_TYP_NEXP} \\[10pt] \frac{\Delta, E \vdash typ \rightsquigarrow t}{\Delta, E \vdash (typ) \, l \rightsquigarrow t} \quad \text{CONVERT_TYP_PAREN} \\[10pt] \frac{\Delta, E \vdash typ \rightsquigarrow t_1 \quad \Delta \vdash t_1 = t_2}{\Delta, E \vdash typ \rightsquigarrow t_2} \quad \text{CONVERT_TYP_EQ} \end{array}$$

$\boxed{\vdash Nexp \rightsquigarrow ne}$ Convert and normalize numeric expressions

$$\frac{}{\vdash N \, l \rightsquigarrow N} \quad \text{CONVERT_NEXP_VAR}$$

$$\frac{}{\vdash num \rightsquigarrow num} \quad \text{CONVERT_NEXP_NUM}$$

$$\frac{}{\vdash num * N \rightsquigarrow num * N} \quad \text{CONVERT_NEXP_MULT}$$

$$\frac{\begin{array}{c} \vdash Nexp_1 \rightsquigarrow ne_1 \\ \vdash Nexp_2 \rightsquigarrow ne_2 \end{array}}{\vdash Nexp_1 + Nexp_2 \rightsquigarrow ne_1 + ne_2} \quad \text{CONVERT_NEXP_ADD}$$

$$\boxed{\Delta, E \vdash \text{typs} \rightsquigarrow t_multi}$$

$$\frac{\Delta, E \vdash \text{typ}_1 \rightsquigarrow t_1 \quad \dots \quad \Delta, E \vdash \text{typ}_n \rightsquigarrow t_n}{\Delta, E \vdash \text{typ}_1 * \dots * \text{typ}_n \rightsquigarrow (t_1 * \dots * t_n)} \quad \text{CONVERT_TYP_ALL}$$

$$\boxed{\vdash \text{lit} : t} \quad \text{Typing literal constants}$$

$$\frac{}{\vdash \text{true } l : _ \text{bool}} \quad \text{CHECK_LIT_TRUE}$$

$$\frac{}{\vdash \text{false } l : _ \text{bool}} \quad \text{CHECK_LIT_FALSE}$$

$$\frac{}{\vdash \text{num } l : _ \text{num}} \quad \text{CHECK_LIT_NUM}$$

$$\frac{num = \text{bitlength}(hex)}{\vdash hex \, l : _ \text{vector } num \, _ \text{bit}} \quad \text{CHECK_LIT_HEX}$$

$$\frac{num = \text{bitlength}(bin)}{\vdash bin \, l : _ \text{vector } num \, _ \text{bit}} \quad \text{CHECK_LIT_BIN}$$

$$\frac{}{\vdash \text{string } l : _ \text{string}} \quad \text{CHECK_LIT_STRING}$$

$$\frac{}{\vdash () \, l : _ \text{unit}} \quad \text{CHECK_LIT_UNIT}$$

$$\frac{}{\vdash \text{bitzero } l : _ \text{bit}} \quad \text{CHECK_LIT_BITZERO}$$

$$\frac{}{\vdash \text{bitone } l : _ \text{bit}} \quad \text{CHECK_LIT_BITONE}$$

$$\boxed{\Delta, E \vdash \text{field } id : p \, t_args \rightarrow t \triangleright (x \text{ of } names)} \quad \text{Field typing (also returns canonical field names)}$$

$$\begin{array}{l} E(\overline{x_i^l}^i) \triangleright \langle E^M, E^P, E^F, E^X \rangle \\ E^F(y) \triangleright \langle \text{forall } tnv_1 \dots tnv_n. p \rightarrow t, (z \text{ of } names) \rangle \\ \Delta \vdash t_1 \text{ ok} \quad \dots \quad \Delta \vdash t_n \text{ ok} \end{array}$$

$$\frac{}{\Delta, E \vdash \text{field } \overline{x_i^l}^i \, y \, l_1 \, l_2 : p \, t_1 \dots t_n \rightarrow \{tnv_1 \mapsto t_1 \dots tnv_n \mapsto t_n\}(t) \triangleright (z \text{ of } names)} \quad \text{INST_FIELD_ALL}$$

$$\boxed{\Delta, E \vdash \text{ctor } id : t_multi \rightarrow p \, t_args \triangleright (x \text{ of } names)} \quad \text{Data constructor typing (also returns canonical constructors)}$$

$$\begin{array}{l} E(\overline{x_i^l}^i) \triangleright \langle E^M, E^P, E^F, E^X \rangle \\ E^X(y) \triangleright \langle \text{forall } tnv_1 \dots tnv_n. t_multi \rightarrow p, (z \text{ of } names) \rangle \\ \Delta \vdash t_1 \text{ ok} \quad \dots \quad \Delta \vdash t_n \text{ ok} \end{array}$$

$$\frac{}{\Delta, E \vdash \text{ctor } \overline{x_i^l}^i \, y \, l_1 \, l_2 : \{tnv_1 \mapsto t_1 \dots tnv_n \mapsto t_n\}(t_multi) \rightarrow p \, t_1 \dots t_n \triangleright (z \text{ of } names)} \quad \text{INST_CTOR_ALL}$$

$$\boxed{\Delta, E \vdash \text{val } id : t \triangleright \Sigma^C} \quad \text{Typing top-level bindings, collecting typeclass constraints}$$

$$\begin{array}{c}
\frac{
\begin{array}{l}
E(\overline{x_i^l})^i \triangleright \langle E^M, E^P, E^F, E^X \rangle \\
E^X(y) \triangleright \langle \mathbf{forall} \, tnv_1 \dots tnv_n. (p_1 \, tnv'_1) \dots (p_i \, tnv'_i) \Rightarrow t, env_tag \rangle \\
\Delta \vdash t_1 \mathbf{ok} \quad \dots \quad \Delta \vdash t_n \mathbf{ok} \\
\sigma = \{ tnv_1 \mapsto t_1 \dots tnv_n \mapsto t_n \}
\end{array}
}{
\Delta, E \vdash \mathbf{val} \, \overline{x_i^l}^i \, y \, l_1 \, l_2 : \sigma(t) \triangleright \{ (p_1 \, \sigma(tnv'_1)), \dots, (p_i \, \sigma(tnv'_i)) \}
} \quad \text{INST_VAL_ALL}
\\
\boxed{E, E^L \vdash x \mathbf{not ctor}} \quad v \text{ is not bound to a data constructor}
\\
\frac{E^L(x) \triangleright t}{E, E^L \vdash x \mathbf{not ctor}} \quad \text{NOT_CTOR_VAL}
\\
\frac{x \notin \mathbf{dom}(E^X)}{\langle E^M, E^P, E^F, E^X \rangle, E^L \vdash x \mathbf{not ctor}} \quad \text{NOT_CTOR_UNBOUND}
\\
\frac{E^X(x) \triangleright \langle \mathbf{forall} \, tnv_1 \dots tnv_n. (p_1 \, tnv'_1) \dots (p_i \, tnv'_i) \Rightarrow t, env_tag \rangle}{\langle E^M, E^P, E^F, E^X \rangle, E^L \vdash x \mathbf{not ctor}} \quad \text{NOT_CTOR_BOUND}
\\
\boxed{E^L \vdash id \mathbf{not shadowed}} \quad id \text{ is not lexically shadowed}
\\
\frac{x \notin \mathbf{dom}(E^L)}{E^L \vdash x \, l_1 \, l_2 \mathbf{not shadowed}} \quad \text{NOT_SHADOWED_SING}
\\
\frac{}{E^L \vdash x_1^l \dots x_n^l. y^l. z^l \, l \mathbf{not shadowed}} \quad \text{NOT_SHADOWED_MULTI}
\\
\boxed{\Delta, E, E_1^L \vdash pat : t \triangleright E_2^L} \quad \text{Typing patterns, building their binding environment}
\\
\frac{\Delta, E, E_1^L \vdash pat_aux : t \triangleright E_2^L}{\Delta, E, E_1^L \vdash pat_aux \, l : t \triangleright E_2^L} \quad \text{CHECK_PAT_ALL}
\\
\boxed{\Delta, E, E_1^L \vdash pat_aux : t \triangleright E_2^L} \quad \text{Typing patterns, building their binding environment}
\\
\frac{\Delta \vdash t \mathbf{ok}}{\Delta, E, E^L \vdash _ : t \triangleright \{ \}} \quad \text{CHECK_PAT_AUX_WILD}
\\
\frac{
\begin{array}{l}
\Delta, E, E_1^L \vdash pat : t \triangleright E_2^L \\
x \notin \mathbf{dom}(E_2^L)
\end{array}
}{\Delta, E, E_1^L \vdash (pat \mathbf{as} \, x \, l) : t \triangleright E_2^L \uplus \{ x \mapsto t \}} \quad \text{CHECK_PAT_AUX_AS}
\\
\frac{
\begin{array}{l}
\Delta, E, E_1^L \vdash pat : t \triangleright E_2^L \\
\Delta, E \vdash typ \rightsquigarrow t
\end{array}
}{\Delta, E, E_1^L \vdash (pat : typ) : t \triangleright E_2^L} \quad \text{CHECK_PAT_AUX_TYP}
\\
\frac{
\begin{array}{l}
\Delta, E \vdash \mathbf{ctor} \, id : (t_1 * \dots * t_n) \rightarrow p \, t_args \triangleright (x \mathbf{of} \, names) \\
E^L \vdash id \mathbf{not shadowed} \\
\Delta, E, E^L \vdash pat_1 : t_1 \triangleright E_1^L \quad \dots \quad \Delta, E, E^L \vdash pat_n : t_n \triangleright E_n^L \\
\mathbf{disjoint doms}(E_1^L, \dots, E_n^L)
\end{array}
}{\Delta, E, E^L \vdash id \, pat_1 \dots pat_n : p \, t_args \triangleright E_1^L \uplus \dots \uplus E_n^L} \quad \text{CHECK_PAT_AUX_IDENT_CONSTR}
\\
\frac{
\begin{array}{l}
\Delta \vdash t \mathbf{ok} \\
E, E^L \vdash x \mathbf{not ctor}
\end{array}
}{\Delta, E, E^L \vdash x \, l_1 \, l_2 : t \triangleright \{ x \mapsto t \}} \quad \text{CHECK_PAT_AUX_VAR}
\end{array}$$

$$\begin{array}{c}
\frac{\Delta, E \vdash \mathbf{field} \, id_i : p \, t_args \rightarrow t_i \triangleright (x_i \mathbf{of} \, names)^i}{\Delta, E, E^L \vdash pat_i : t_i \triangleright E_i^L} \\
\frac{\mathbf{disjoint \, doms}(\overline{E_i^L}^i) \quad \mathbf{duplicates}(\overline{x_i}^i) = \emptyset}{\Delta, E, E^L \vdash \langle | \overline{id_i} = pat_i \, \overline{l_i}^i ; ? | \rangle : p \, t_args \triangleright \uplus \overline{E_i^L}^i} \text{CHECK_PAT_AUX_RECORD} \\
\frac{\Delta, E, E^L \vdash pat_1 : t \triangleright E_1^L \quad \dots \quad \Delta, E, E^L \vdash pat_n : t \triangleright E_n^L \quad \mathbf{disjoint \, doms}(E_1^L, \dots, E_n^L) \quad \mathbf{length}(pat_1 \dots pat_n) = num}{\Delta, E, E^L \vdash [pat_1; \dots; pat_n] : \mathbf{--vector} \, num \, t \triangleright E_1^L \uplus \dots \uplus E_n^L} \text{CHECK_PAT_AUX_VECTOR} \\
\frac{\Delta, E, E^L \vdash pat_1 : \mathbf{--vector} \, ne_1 \, t \triangleright E_1^L \quad \dots \quad \Delta, E, E^L \vdash pat_n : \mathbf{--vector} \, ne_n \, t \triangleright E_n^L \quad \mathbf{disjoint \, doms}(E_1^L, \dots, E_n^L) \quad ne' = ne_1 + \dots + ne_n}{\Delta, E, E^L \vdash [pat_1 \dots pat_n] : \mathbf{--vector} \, ne' \, t \triangleright E_1^L \uplus \dots \uplus E_n^L} \text{CHECK_PAT_AUX_VECTOR} \\
\frac{\Delta, E, E^L \vdash pat_1 : t_1 \triangleright E_1^L \quad \dots \quad \Delta, E, E^L \vdash pat_n : t_n \triangleright E_n^L \quad \mathbf{disjoint \, doms}(E_1^L, \dots, E_n^L)}{\Delta, E, E^L \vdash (pat_1, \dots, pat_n) : t_1 * \dots * t_n \triangleright E_1^L \uplus \dots \uplus E_n^L} \text{CHECK_PAT_AUX_TUP} \\
\frac{\Delta \vdash t \mathbf{ok} \quad \Delta, E, E^L \vdash pat_1 : t \triangleright E_1^L \quad \dots \quad \Delta, E, E^L \vdash pat_n : t \triangleright E_n^L \quad \mathbf{disjoint \, doms}(E_1^L, \dots, E_n^L)}{\Delta, E, E^L \vdash [pat_1; \dots; pat_n; ?] : \mathbf{--list} \, t \triangleright E_1^L \uplus \dots \uplus E_n^L} \text{CHECK_PAT_AUX_LIST} \\
\frac{\Delta, E, E_1^L \vdash pat : t \triangleright E_2^L}{\Delta, E, E_1^L \vdash (pat) : t \triangleright E_2^L} \text{CHECK_PAT_AUX_PAREN} \\
\frac{\Delta, E, E_1^L \vdash pat_1 : t \triangleright E_2^L \quad \Delta, E, E_1^L \vdash pat_2 : \mathbf{--list} \, t \triangleright E_3^L \quad \mathbf{disjoint \, doms}(E_2^L, E_3^L)}{\Delta, E, E_1^L \vdash pat_1 :: pat_2 : \mathbf{--list} \, t \triangleright E_2^L \uplus E_3^L} \text{CHECK_PAT_AUX_CONS} \\
\frac{\vdash lit : t}{\Delta, E, E^L \vdash lit : t \triangleright \{ \}} \text{CHECK_PAT_AUX_LIT} \\
\frac{E, E^L \vdash x \mathbf{not \, ctor}}{\Delta, E, E^L \vdash x \, l + num : \mathbf{--num} \triangleright \{ x \mapsto \mathbf{--num} \}} \text{CHECK_PAT_AUX_NUM_ADD}
\end{array}$$

$E \vdash id \mathbf{field}$

Check that the identifier is a permissible field identifier

$$\begin{array}{c}
\frac{E^F(x) \triangleright f_desc}{\langle E^M, E^P, E^F, E^X \rangle \vdash x \, l_1 \, l_2 \mathbf{field}} \text{ID_FIELD_EMPTY} \\
\frac{E^M(x) \triangleright E \quad x \notin \mathbf{dom}(E^F) \quad E \vdash \overline{y_i^L}^i \, z^l \, l_2 \mathbf{field}}{\langle E^M, E^P, E^F, E^X \rangle \vdash x \, l_1. \overline{y_i^L}^i \, z^l \, l_2 \mathbf{field}} \text{ID_FIELD_CONS}
\end{array}$$

$E \vdash id \mathbf{value}$

Check that the identifier is a permissible value identifier

$$\frac{E^X(x) \triangleright v_desc}{\langle E^M, E^P, E^F, E^X \rangle \vdash x \, l_1 \, l_2 \mathbf{value}} \text{ID_VALUE_EMPTY}$$

$$\begin{array}{c}
\frac{
\begin{array}{l}
E^M(x) \triangleright E \\
x \notin \mathbf{dom}(E^X) \\
E \vdash \overline{y_i^l}^i z^l l_2 \mathbf{value}
\end{array}
}{
\langle E^M, E^P, E^F, E^X \rangle \vdash x l_1. \overline{y_i^l}^i z^l l_2 \mathbf{value}
} \text{ID_VALUE_CONS}
\\
\boxed{\Delta, E, E^L \vdash \mathit{exp} : t \triangleright \Sigma^C, \Sigma^N} \quad \text{Typing expressions, collecting typeclass and index constraints}
\\
\frac{
\Delta, E, E^L \vdash \mathit{exp_aux} : t \triangleright \Sigma^C, \Sigma^N
}{
\Delta, E, E^L \vdash \mathit{exp_aux} l : t \triangleright \Sigma^C, \Sigma^N
} \text{CHECK_EXP_ALL}
\\
\boxed{\Delta, E, E^L \vdash \mathit{exp_aux} : t \triangleright \Sigma^C, \Sigma^N} \quad \text{Typing expressions, collecting typeclass and index constraints}
\\
\frac{
E^L(x) \triangleright t
}{
\Delta, E, E^L \vdash x l_1 l_2 : t \triangleright \{\}, \{\}
} \text{CHECK_EXP_AUX_VAR}
\\
\frac{
}{
\Delta, E, E^L \vdash N : \mathit{num} \triangleright \{\}, \{\}
} \text{CHECK_EXP_AUX_NVAR}
\\
\begin{array}{l}
E^L \vdash \mathit{id} \mathbf{not shadowed} \\
E \vdash \mathit{id} \mathbf{value} \\
\Delta, E \vdash \mathbf{ctor} \mathit{id} : t_multi \rightarrow p t_args \triangleright (x \mathbf{of} names)
\end{array}
\\
\frac{
}{
\Delta, E, E^L \vdash \mathit{id} : \mathbf{curry} (t_multi, p t_args) \triangleright \{\}, \{\}
} \text{CHECK_EXP_AUX_CTOR}
\\
\begin{array}{l}
E^L \vdash \mathit{id} \mathbf{not shadowed} \\
E \vdash \mathit{id} \mathbf{value} \\
\Delta, E \vdash \mathbf{val} \mathit{id} : t \triangleright \Sigma^C
\end{array}
\\
\frac{
}{
\Delta, E, E^L \vdash \mathit{id} : t \triangleright \Sigma^C, \{\}
} \text{CHECK_EXP_AUX_VAL}
\\
\begin{array}{l}
\Delta, E, E^L \vdash \mathit{pat}_1 : t_1 \triangleright E_1^L \quad \dots \quad \Delta, E, E^L \vdash \mathit{pat}_n : t_n \triangleright E_n^L \\
\Delta, E, E^L \uplus E_1^L \uplus \dots \uplus E_n^L \vdash \mathit{exp} : u \triangleright \Sigma^C, \Sigma^N \\
\mathbf{disjoint doms}(E_1^L, \dots, E_n^L)
\end{array}
\\
\frac{
}{
\Delta, E, E^L \vdash \mathbf{fun} \mathit{pat}_1 \dots \mathit{pat}_n \rightarrow \mathit{exp} l : \mathbf{curry} ((t_1 * \dots * t_n), u) \triangleright \Sigma^C, \Sigma^N
} \text{CHECK_EXP_AUX_FN}
\\
\frac{
\Delta, E, E^L \vdash \mathit{pat}_i : t \triangleright E_i^L
}{
\Delta, E, E^L \uplus E_i^L \vdash \mathit{exp}_i : u \triangleright \Sigma^C_i, \Sigma^N_i
}
\\
\frac{
}{
\Delta, E, E^L \vdash \mathbf{function} [^? \overline{\mathit{pat}_i \rightarrow \mathit{exp}_i l_i}^i \mathbf{end} : t \rightarrow u \triangleright \overline{\Sigma^C_i}^i, \overline{\Sigma^N_i}^i
} \text{CHECK_EXP_AUX_FUNCTION}
\\
\begin{array}{l}
\Delta, E, E^L \vdash \mathit{exp}_1 : t_1 \rightarrow t_2 \triangleright \Sigma^C_1, \Sigma^N_1 \\
\Delta, E, E^L \vdash \mathit{exp}_2 : t_1 \triangleright \Sigma^C_2, \Sigma^N_2
\end{array}
\\
\frac{
}{
\Delta, E, E^L \vdash \mathit{exp}_1 \mathit{exp}_2 : t_2 \triangleright \Sigma^C_1 \cup \Sigma^C_2, \Sigma^N_1 \cup \Sigma^N_2
} \text{CHECK_EXP_AUX_APP}
\\
\begin{array}{l}
\Delta, E, E^L \vdash (ix) : t_1 \rightarrow t_2 \rightarrow t_3 \triangleright \Sigma^C_1, \Sigma^N_1 \\
\Delta, E, E^L \vdash \mathit{exp}_1 : t_1 \triangleright \Sigma^C_2, \Sigma^N_2 \\
\Delta, E, E^L \vdash \mathit{exp}_2 : t_2 \triangleright \Sigma^C_3, \Sigma^N_3
\end{array}
\\
\frac{
}{
\Delta, E, E^L \vdash \mathit{exp}_1 ix l \mathit{exp}_2 : t_3 \triangleright \Sigma^C_1 \cup \Sigma^C_2 \cup \Sigma^C_3, \Sigma^N_1 \cup \Sigma^N_2 \cup \Sigma^N_3
} \text{CHECK_EXP_AUX_INFIX_APP1}
\\
\begin{array}{l}
\Delta, E, E^L \vdash x : t_1 \rightarrow t_2 \rightarrow t_3 \triangleright \Sigma^C_1, \Sigma^N_1 \\
\Delta, E, E^L \vdash \mathit{exp}_1 : t_1 \triangleright \Sigma^C_2, \Sigma^N_2 \\
\Delta, E, E^L \vdash \mathit{exp}_2 : t_2 \triangleright \Sigma^C_3, \Sigma^N_3
\end{array}
\\
\frac{
}{
\Delta, E, E^L \vdash \mathit{exp}_1 'x' l \mathit{exp}_2 : t_3 \triangleright \Sigma^C_1 \cup \Sigma^C_2 \cup \Sigma^C_3, \Sigma^N_1 \cup \Sigma^N_2 \cup \Sigma^N_3
} \text{CHECK_EXP_AUX_INFIX_APP2}
\\
\begin{array}{l}
\Delta, E \vdash \mathbf{field} \mathit{id}_i : p t_args \rightarrow t_i \triangleright (x_i \mathbf{of} names)^i \\
\Delta, E, E^L \vdash \mathit{exp}_i : t_i \triangleright \Sigma^C_i, \Sigma^N_i
\end{array}
\\
\begin{array}{l}
\mathbf{duplicates}(\overline{x_i}^i) = \emptyset \\
names = \{\overline{x_i}^i\}
\end{array}
\\
\frac{
}{
\Delta, E, E^L \vdash \langle \overline{\mathit{id}_i = \mathit{exp}_i l_i}^i, ? l \rangle : p t_args \triangleright \overline{\Sigma^C_i}^i, \overline{\Sigma^N_i}^i
} \text{CHECK_EXP_AUX_RECORD}
\end{array}$$

$$\begin{array}{c}
\frac{\Delta, E \vdash \mathbf{field} \, id_i : p \, t_args \rightarrow t_i \triangleright (x_i \mathbf{of} \, names)^i}{\Delta, E, E^L \vdash exp_i : t_i \triangleright \Sigma^C_i, \Sigma^N_i} \\
\frac{\mathbf{duplicates}(\overline{x_i}^i) = \emptyset}{\Delta, E, E^L \vdash exp : p \, t_args \triangleright \Sigma^{C'}, \Sigma^{N'}} \\
\hline
\Delta, E, E^L \vdash \langle |exp \mathbf{with} \overline{id_i = exp_i \, l_i^i; ? \, l}| \rangle : p \, t_args \triangleright \Sigma^{C'} \cup \overline{\Sigma^C_i}^i, \Sigma^{N'} \cup \overline{\Sigma^N_i}^i \quad \text{CHECK_EXP_AUX_RECUP} \\
\Delta, E, E^L \vdash exp_1 : t \triangleright \Sigma^C_1, \Sigma^N_1 \quad \dots \quad \Delta, E, E^L \vdash exp_n : t \triangleright \Sigma^C_n, \Sigma^N_n \\
\mathbf{length}(exp_1 \dots exp_n) = num \\
\hline
\Delta, E, E^L \vdash [|exp_1; \dots; exp_n|] : \mathbf{vector} \, num \, t \triangleright \Sigma^C_1 \cup \dots \cup \Sigma^C_n, \Sigma^N_1 \cup \dots \cup \Sigma^N_n \quad \text{CHECK_EXP_AUX_VECTOR} \\
\Delta, E, E^L \vdash exp : \mathbf{vector} \, ne' \, t \triangleright \Sigma^C, \Sigma^N \\
\vdash Nex \rightsquigarrow ne \\
\hline
\Delta, E, E^L \vdash exp.(Nex) : t \triangleright \Sigma^C, \Sigma^N \cup \{ne \langle ne' \rangle\} \quad \text{CHECK_EXP_AUX_VECTORGET} \\
\Delta, E, E^L \vdash exp : \mathbf{vector} \, ne' \, t \triangleright \Sigma^C, \Sigma^N \\
\vdash Nex_1 \rightsquigarrow ne_1 \\
\vdash Nex_2 \rightsquigarrow ne_2 \\
ne = ne_2 + (-ne_1) \\
\hline
\Delta, E, E^L \vdash exp.(Nex_1..Nex_2) : \mathbf{vector} \, ne \, t \triangleright \Sigma^C, \Sigma^N \cup \{ne_1 \langle ne_2 \rangle\} \quad \text{CHECK_EXP_AUX_VECTORSUB} \\
E \vdash id \mathbf{field} \\
\Delta, E \vdash \mathbf{field} \, id : p \, t_args \rightarrow t \triangleright (x \mathbf{of} \, names) \\
\Delta, E, E^L \vdash exp : p \, t_args \triangleright \Sigma^C, \Sigma^N \\
\hline
\Delta, E, E^L \vdash exp.id : t \triangleright \Sigma^C, \Sigma^N \quad \text{CHECK_EXP_AUX_FIELD} \\
\Delta, E, E^L \vdash pat_i : t \triangleright E_i^L{}^i \\
\Delta, E, E^L \uplus E_i^L \vdash exp_i : u \triangleright \Sigma^C_i, \Sigma^N_i{}^i \\
\Delta, E, E^L \vdash exp : t \triangleright \Sigma^{C'}, \Sigma^{N'} \\
\hline
\Delta, E, E^L \vdash \mathbf{match} \, exp \mathbf{with} |? \, pat_i \rightarrow exp_i \, \overline{l_i}^i \mathbf{end} : u \triangleright \Sigma^{C'} \cup \overline{\Sigma^C_i}^i, \Sigma^{N'} \cup \overline{\Sigma^N_i}^i \quad \text{CHECK_EXP_AUX_CASE} \\
\Delta, E, E^L \vdash exp : t \triangleright \Sigma^C, \Sigma^N \\
\Delta, E \vdash typ \rightsquigarrow t \\
\hline
\Delta, E, E^L \vdash (exp : typ) : t \triangleright \Sigma^C, \Sigma^N \quad \text{CHECK_EXP_AUX_TYPED} \\
\Delta, E, E_1^L \vdash letbind \triangleright E_2^L, \Sigma^C_1, \Sigma^N_1 \\
\Delta, E, E_1^L \uplus E_2^L \vdash exp : t \triangleright \Sigma^C_2, \Sigma^N_2 \\
\hline
\Delta, E, E^L \vdash \mathbf{let} \, letbind \mathbf{in} \, exp : t \triangleright \Sigma^C_1 \cup \Sigma^C_2, \Sigma^N_1 \cup \Sigma^N_2 \quad \text{CHECK_EXP_AUX_LET} \\
\Delta, E, E^L \vdash exp_1 : t_1 \triangleright \Sigma^C_1, \Sigma^N_1 \quad \dots \quad \Delta, E, E^L \vdash exp_n : t_n \triangleright \Sigma^C_n, \Sigma^N_n \\
\hline
\Delta, E, E^L \vdash (exp_1, \dots, exp_n) : t_1 * \dots * t_n \triangleright \Sigma^C_1 \cup \dots \cup \Sigma^C_n, \Sigma^N_1 \cup \dots \cup \Sigma^N_n \quad \text{CHECK_EXP_AUX_TUP} \\
\Delta \vdash t \mathbf{ok} \\
\Delta, E, E^L \vdash exp_1 : t \triangleright \Sigma^C_1, \Sigma^N_1 \quad \dots \quad \Delta, E, E^L \vdash exp_n : t \triangleright \Sigma^C_n, \Sigma^N_n \\
\hline
\Delta, E, E^L \vdash [exp_1; \dots; exp_n; ?] : \mathbf{list} \, t \triangleright \Sigma^C_1 \cup \dots \cup \Sigma^C_n, \Sigma^N_1 \cup \dots \cup \Sigma^N_n \quad \text{CHECK_EXP_AUX_LIST} \\
\Delta, E, E^L \vdash exp : t \triangleright \Sigma^C, \Sigma^N \\
\hline
\Delta, E, E^L \vdash (exp) : t \triangleright \Sigma^C, \Sigma^N \quad \text{CHECK_EXP_AUX_PAREN} \\
\Delta, E, E^L \vdash exp : t \triangleright \Sigma^C, \Sigma^N \\
\hline
\Delta, E, E^L \vdash \mathbf{begin} \, exp \mathbf{end} : t \triangleright \Sigma^C, \Sigma^N \quad \text{CHECK_EXP_AUX_BEGIN} \\
\Delta, E, E^L \vdash exp_1 : \mathbf{bool} \triangleright \Sigma^C_1, \Sigma^N_1 \\
\Delta, E, E^L \vdash exp_2 : t \triangleright \Sigma^C_2, \Sigma^N_2 \\
\Delta, E, E^L \vdash exp_3 : t \triangleright \Sigma^C_3, \Sigma^N_3 \\
\hline
\Delta, E, E^L \vdash \mathbf{if} \, exp_1 \mathbf{then} \, exp_2 \mathbf{else} \, exp_3 : t \triangleright \Sigma^C_1 \cup \Sigma^C_2 \cup \Sigma^C_3, \Sigma^N_1 \cup \Sigma^N_2 \cup \Sigma^N_3 \quad \text{CHECK_EXP_AUX_IF}
\end{array}$$

$$\begin{array}{c}
\frac{\Delta, E, E^L \vdash \text{exp}_1 : t \triangleright \Sigma^C_1, \Sigma^N_1 \quad \Delta, E, E^L \vdash \text{exp}_2 : _ \text{list } t \triangleright \Sigma^C_2, \Sigma^N_2}{\Delta, E, E^L \vdash \text{exp}_1 :: \text{exp}_2 : _ \text{list } t \triangleright \Sigma^C_1 \cup \Sigma^C_2, \Sigma^N_1 \cup \Sigma^N_2} \text{CHECK_EXP_AUX_CONS} \\
\\
\frac{\vdash \text{lit} : t}{\Delta, E, E^L \vdash \text{lit} : t \triangleright \{\}, \{\}} \text{CHECK_EXP_AUX_LIT} \\
\\
\frac{\overline{\Delta \vdash t_i \text{ok}}^i \quad \Delta, E, E^L \uplus \{ \overline{x_i \mapsto t_i}^i \} \vdash \text{exp}_1 : t \triangleright \Sigma^C_1, \Sigma^N_1 \quad \Delta, E, E^L \uplus \{ \overline{x_i \mapsto t_i}^i \} \vdash \text{exp}_2 : _ \text{bool} \triangleright \Sigma^C_2, \Sigma^N_2 \quad \text{disjoint doms}(E^L, \{ \overline{x_i \mapsto t_i}^i \}) \quad E = \langle E^M, E^P, E^F, E^X \rangle}{\overline{x_i \notin \text{dom}(E^X)}^i} \text{CHECK_EXP_AUX_SET_COMP} \\
\\
\frac{\Delta, E, E^L \vdash \{ \text{exp}_1 | \text{exp}_2 \} : _ \text{set } t \triangleright \Sigma^C_1 \cup \Sigma^C_2, \Sigma^N_1 \cup \Sigma^N_2}{\Delta, E, E^L \vdash \{ \text{exp}_1 | \text{forall } \overline{qbind_i}^i | \text{exp}_2 \} : _ \text{set } t \triangleright \Sigma^C_1 \cup \Sigma^C_2 \cup \Sigma^C_3, \Sigma^N_2 \cup \Sigma^N_3} \text{CHECK_EXP_AUX_SET_COMP} \\
\\
\frac{\Delta \vdash t \text{ok} \quad \Delta, E, E^L \vdash \text{exp}_1 : t \triangleright \Sigma^C_1, \Sigma^N_1 \quad \dots \quad \Delta, E, E^L \vdash \text{exp}_n : t \triangleright \Sigma^C_n, \Sigma^N_n}{\Delta, E, E^L \vdash \{ \text{exp}_1; \dots; \text{exp}_n; ? \} : _ \text{set } t \triangleright \Sigma^C_1 \cup \dots \cup \Sigma^C_n, \Sigma^N_1 \cup \dots \cup \Sigma^N_n} \text{CHECK_EXP_AUX_SET} \\
\\
\frac{\Delta, E, E^L \vdash \overline{qbind_i}^i \triangleright E^L_2, \Sigma^C_1 \quad \Delta, E, E^L \uplus E^L_2 \vdash \text{exp} : _ \text{bool} \triangleright \Sigma^C_2, \Sigma^N_2}{\Delta, E, E^L \vdash q \overline{qbind_i}^i . \text{exp} : _ \text{bool} \triangleright \Sigma^C_1 \cup \Sigma^C_2, \Sigma^N_2} \text{CHECK_EXP_AUX_QUANT} \\
\\
\frac{\Delta, E, E^L \vdash \text{list } \overline{qbind_i}^i \triangleright E^L_2, \Sigma^C_1 \quad \Delta, E, E^L \uplus E^L_2 \vdash \text{exp}_1 : t \triangleright \Sigma^C_2, \Sigma^N_2 \quad \Delta, E, E^L \uplus E^L_2 \vdash \text{exp}_2 : _ \text{bool} \triangleright \Sigma^C_3, \Sigma^N_3}{\Delta, E, E^L \vdash [\text{exp}_1 | \text{forall } \overline{qbind_i}^i | \text{exp}_2] : _ \text{list } t \triangleright \Sigma^C_1 \cup \Sigma^C_2 \cup \Sigma^C_3, \Sigma^N_2 \cup \Sigma^N_3} \text{CHECK_EXP_AUX_LIST_COMP} \\
\\
\boxed{\Delta, E, E^L \vdash qbind_1 \dots qbind_n \triangleright E^L_2, \Sigma^C} \quad \text{Build the environment for quantifier bindings, collecting typeclass constraints} \\
\\
\frac{}{\overline{\Delta, E, E^L \vdash \triangleright \{\}, \{\}}} \text{CHECK_LISTQUANT_BINDING_EMPTY} \\
\\
\frac{\Delta \vdash t \text{ok} \quad \Delta, E, E^L \uplus \{ x \mapsto t \} \vdash \overline{qbind_i}^i \triangleright E^L_2, \Sigma^C_1 \quad \text{disjoint doms}(\{ x \mapsto t \}, E^L_2)}{\Delta, E, E^L \vdash x \text{ l } \overline{qbind_i}^i \triangleright \{ x \mapsto t \} \uplus E^L_2, \Sigma^C_1} \text{CHECK_LISTQUANT_BINDING_VAR} \\
\\
\frac{\Delta, E, E^L \vdash \text{pat} : t \triangleright E^L_3 \quad \Delta, E, E^L \vdash \text{exp} : _ \text{set } t \triangleright \Sigma^C_1, \Sigma^N_1 \quad \Delta, E, E^L \uplus E^L_3 \vdash \overline{qbind_i}^i \triangleright E^L_2, \Sigma^C_2 \quad \text{disjoint doms}(E^L_3, E^L_2)}{\Delta, E, E^L \vdash (\text{pat IN exp}) \overline{qbind_i}^i \triangleright E^L_2 \uplus E^L_3, \Sigma^C_1 \cup \Sigma^C_2} \text{CHECK_LISTQUANT_BINDING_RESTR} \\
\\
\frac{\Delta, E, E^L \vdash \text{pat} : t \triangleright E^L_3 \quad \Delta, E, E^L \vdash \text{exp} : _ \text{list } t \triangleright \Sigma^C_1, \Sigma^N_1 \quad \Delta, E, E^L \uplus E^L_3 \vdash \overline{qbind_i}^i \triangleright E^L_2, \Sigma^C_2 \quad \text{disjoint doms}(E^L_3, E^L_2)}{\Delta, E, E^L \vdash (\text{pat MEM exp}) \overline{qbind_i}^i \triangleright E^L_2 \uplus E^L_3, \Sigma^C_1 \cup \Sigma^C_2} \text{CHECK_LISTQUANT_BINDING_LIST_RESTR}
\end{array}$$

$\Delta, E, E_1^L \vdash \mathbf{list} \ qbind_1 .. qbind_n \triangleright E_2^L, \Sigma^C$	Build the environment for quantifier bindings, collecting typeclass
$\frac{\Delta, E, E^L \vdash \mathbf{list} \triangleright \{\}, \{\}}{\text{CHECK_QUANT_BINDING_EMPTY}}$	
$\frac{\begin{array}{l} \Delta, E, E_1^L \vdash pat : t \triangleright E_3^L \\ \Delta, E, E_1^L \vdash exp : \mathbf{list} \ t \triangleright \Sigma^C_1, \Sigma^N_1 \\ \Delta, E, E_1^L \uplus E_3^L \vdash \overline{qbind_i}^i \triangleright E_2^L, \Sigma^C_2 \\ \mathbf{disjoint\ doms} (E_3^L, E_2^L) \end{array}}{\Delta, E, E_1^L \vdash \mathbf{list} (pat \mathbf{MEM} exp) \overline{qbind_i}^i \triangleright E_2^L \uplus E_3^L, \Sigma^C_1 \cup \Sigma^C_2}$	CHECK_QUANT_BINDING_RESTR
$\Delta, E, E^L \vdash funcl \triangleright \{x \mapsto t\}, \Sigma^C, \Sigma^N$	Build the environment for a function definition clause, collecting typeclass
$\frac{\begin{array}{l} \Delta, E, E^L \vdash pat_1 : t_1 \triangleright E_1^L \quad \dots \quad \Delta, E, E^L \vdash pat_n : t_n \triangleright E_n^L \\ \Delta, E, E^L \uplus E_1^L \uplus \dots \uplus E_n^L \vdash exp : u \triangleright \Sigma^C, \Sigma^N \\ \mathbf{disjoint\ doms} (E_1^L, \dots, E_n^L) \\ \Delta, E \vdash typ \rightsquigarrow u \end{array}}{\Delta, E, E^L \vdash x \ l_1 \ pat_1 \dots pat_n : typ = exp \ l_2 \triangleright \{x \mapsto \mathbf{curry} ((t_1 * \dots * t_n), u)\}, \Sigma^C, \Sigma^N}$	CHECK_FUNCL_ANNOT
$\frac{\begin{array}{l} \Delta, E, E^L \vdash pat_1 : t_1 \triangleright E_1^L \quad \dots \quad \Delta, E, E^L \vdash pat_n : t_n \triangleright E_n^L \\ \Delta, E, E^L \uplus E_1^L \uplus \dots \uplus E_n^L \vdash exp : u \triangleright \Sigma^C, \Sigma^N \\ \mathbf{disjoint\ doms} (E_1^L, \dots, E_n^L) \end{array}}{\Delta, E, E^L \vdash x \ l_1 \ pat_1 \dots pat_n = exp \ l_2 \triangleright \{x \mapsto \mathbf{curry} ((t_1 * \dots * t_n), u)\}, \Sigma^C, \Sigma^N}$	CHECK_FUNCL_NOANNOT
$\Delta, E, E_1^L \vdash letbind \triangleright E_2^L, \Sigma^C, \Sigma^N$	Build the environment for a let binding, collecting typeclass and index con
$\frac{\begin{array}{l} \Delta, E, E_1^L \vdash pat : t \triangleright E_2^L \\ \Delta, E, E_1^L \vdash exp : t \triangleright \Sigma^C, \Sigma^N \\ \Delta, E \vdash typ \rightsquigarrow t \end{array}}{\Delta, E, E_1^L \vdash pat : typ = exp \ l \triangleright E_2^L, \Sigma^C, \Sigma^N}$	CHECK_LETBIND_VAL_ANNOT
$\frac{\begin{array}{l} \Delta, E, E_1^L \vdash pat : t \triangleright E_2^L \\ \Delta, E, E_1^L \vdash exp : t \triangleright \Sigma^C, \Sigma^N \end{array}}{\Delta, E, E_1^L \vdash pat = exp \ l \triangleright E_2^L, \Sigma^C, \Sigma^N}$	CHECK_LETBIND_VAL_NOANNOT
$\frac{\Delta, E, E_1^L \vdash funcl_aux \ l \triangleright \{x \mapsto t\}, \Sigma^C, \Sigma^N}{\Delta, E, E_1^L \vdash funcl_aux \ l \triangleright \{x \mapsto t\}, \Sigma^C, \Sigma^N}$	CHECK_LETBIND_FN
$\Delta, E, E^L \vdash rule \triangleright \{x \mapsto t\}, \Sigma^C, \Sigma^N$	Build the environment for an inductive relation clause, collecting typeclass
$\frac{\begin{array}{l} \overline{\Delta \vdash t_i \mathbf{ok}}^i \\ E_2^L = \{ \overline{y_i \mapsto t_i}^i \} \\ \Delta, E, E_1^L \uplus E_2^L \vdash exp' : \mathbf{bool} \triangleright \Sigma^{C'}, \Sigma^{N'} \\ \Delta, E, E_1^L \uplus E_2^L \vdash exp_1 : u_1 \triangleright \Sigma^C_1, \Sigma^N_1 \quad \dots \quad \Delta, E, E_1^L \uplus E_2^L \vdash exp_n : u_n \triangleright \Sigma^C_n, \Sigma^N_n \end{array}}{\Delta, E, E_1^L \vdash id^? \mathbf{forall} \ \overline{y_i \ l_i}^i . exp' \implies x \ l \ exp_1 .. exp_n \ l' \triangleright \{x \mapsto \mathbf{curry} ((u_1 * \dots * u_n), \mathbf{bool})\}, \Sigma^{C'} \cup \Sigma^C_1 \cup \dots \cup \Sigma^C_n}$	
$xs, \Delta_1, E \vdash \mathbf{tc} \ td \triangleright \Delta_2, E^P$	Extract the type constructor information
$\frac{\begin{array}{l} tnvars^l \rightsquigarrow tnvs \\ \Delta, E \vdash typ \rightsquigarrow t \\ \mathbf{duplicates} (tnvs) = \emptyset \\ \mathbf{FV} (t) \subset tnvs \\ \overline{y_i}^i . x \notin \mathbf{dom} (\Delta) \end{array}}{\overline{y_i}^i . \Delta, E \vdash \mathbf{tc} \ x \ l \ tnvars^l = typ \triangleright \{ \overline{y_i}^i . x \mapsto tnvs . t \}, \{x \mapsto \overline{y_i}^i . x\}}$	CHECK_TEXP_TC_ABBREV

$$\begin{array}{c}
\begin{array}{c}
\text{tnvars}^l \rightsquigarrow \text{tnvs} \\
\mathbf{duplicates}(\text{tnvs}) = \emptyset \\
\overline{y_i}^i x \notin \mathbf{dom}(\Delta)
\end{array} \\
\hline
\overline{y_i}^i, \Delta, E_1 \vdash \mathbf{tc} x l \text{tnvars}^l \triangleright \{ \overline{y_i}^i x \mapsto \text{tnvs} \}, \{ x \mapsto \overline{y_i}^i x \} \quad \text{CHECK_TEXP_TC_ABSTRACT}
\end{array}$$

$$\begin{array}{c}
\begin{array}{c}
\text{tnvars}^l \rightsquigarrow \text{tnvs} \\
\mathbf{duplicates}(\text{tnvs}) = \emptyset \\
\overline{y_i}^i x \notin \mathbf{dom}(\Delta)
\end{array} \\
\hline
\overline{y_i}^i, \Delta_1, E \vdash \mathbf{tc} x l \text{tnvars}^l = \langle |x_1^l : \text{typ}_1; \dots; x_j^l : \text{typ}_j; ?| \rangle \triangleright \{ \overline{y_i}^i x \mapsto \text{tnvs} \}, \{ x \mapsto \overline{y_i}^i x \} \quad \text{CHECK_TEXP_TC_REC}
\end{array}$$

$$\begin{array}{c}
\begin{array}{c}
\text{tnvars}^l \rightsquigarrow \text{tnvs} \\
\mathbf{duplicates}(\text{tnvs}) = \emptyset \\
\overline{y_i}^i x \notin \mathbf{dom}(\Delta)
\end{array} \\
\hline
\overline{y_i}^i, \Delta_1, E \vdash \mathbf{tc} x l \text{tnvars}^l = |^? \text{ctor_def}_1 | \dots | \text{ctor_def}_j \triangleright \{ \overline{y_i}^i x \mapsto \text{tnvs} \}, \{ x \mapsto \overline{y_i}^i x \} \quad \text{CHECK_TEXP_TC_VAR}
\end{array}$$

$$\boxed{xs, \Delta_1, E \vdash \mathbf{tc} td_1 .. td_i \triangleright \Delta_2, E^P} \quad \text{Extract the type constructor information}$$

$$\begin{array}{c}
\overline{xs, \Delta, E \vdash \mathbf{tc} \triangleright \{ \}, \{ \}} \quad \text{CHECK_TEXPS_TC_EMPTY} \\
\begin{array}{c}
xs, \Delta_1, E \vdash \mathbf{tc} td \triangleright \Delta_2, E_2^P \\
xs, \Delta_1 \uplus \Delta_2, E \uplus \langle \{ \}, E_2^P, \{ \}, \{ \} \rangle \vdash \mathbf{tc} \overline{td_i}^i \triangleright \Delta_3, E_3^P \\
\mathbf{dom}(E_2^P) \cap \mathbf{dom}(E_3^P) = \emptyset
\end{array} \\
\hline
xs, \Delta_1, E \vdash \mathbf{tc} td \overline{td_i}^i \triangleright \Delta_2 \uplus \Delta_3, E_2^P \uplus E_3^P \quad \text{CHECK_TEXPS_TC_ABBREV}
\end{array}$$

$$\boxed{\Delta, E \vdash \text{tnvs } p = \text{texp} \triangleright \langle E^F, E^X \rangle} \quad \text{Check a type definition, with its path already resolved}$$

$$\begin{array}{c}
\overline{\Delta, E \vdash \text{tnvs } p = \text{typ} \triangleright \langle \{ \}, \{ \} \rangle} \quad \text{CHECK_TEXP_ABBREV} \\
\begin{array}{c}
\overline{\Delta, E \vdash \text{typ}_i \rightsquigarrow t_i}^i \\
\text{names} = \{ \overline{x_i}^i \} \\
\mathbf{duplicates}(\overline{x_i}^i) = \emptyset \\
\mathbf{FV}(t_i) \subset \text{tnvs}^i \\
E^F = \{ x_i \mapsto \langle \mathbf{forall} \text{tnvs}. p \rightarrow t_i, (x_i \mathbf{of} \text{names}) \rangle \}^i
\end{array} \\
\hline
\Delta, E \vdash \text{tnvs } p = \langle |x_i^l : \text{typ}_i^l; ?| \rangle \triangleright \langle E^F, \{ \} \rangle \quad \text{CHECK_TEXP_REC}
\end{array}$$

$$\begin{array}{c}
\overline{\Delta, E \vdash \text{typs}_i \rightsquigarrow t_multi_i}^i \\
\text{names} = \{ \overline{x_i}^i \} \\
\mathbf{duplicates}(\overline{x_i}^i) = \emptyset \\
\mathbf{FV}(t_multi_i) \subset \text{tnvs}^i \\
E^X = \{ x_i \mapsto \langle \mathbf{forall} \text{tnvs}. t_multi_i \rightarrow p, (x_i \mathbf{of} \text{names}) \rangle \}^i
\end{array}$$

$$\overline{\Delta, E \vdash \text{tnvs } p = |^? \overline{x_i^l \mathbf{of} \text{typs}_i}^i \triangleright \langle \{ \}, E^X \rangle} \quad \text{CHECK_TEXP_VAR}$$

$$\boxed{xs, \Delta, E \vdash td_1 .. td_n \triangleright \langle E^F, E^X \rangle}$$

$$\begin{array}{c}
\overline{\overline{y_i}^i, \Delta, E \vdash \triangleright \langle \{ \}, \{ \} \rangle} \quad \text{CHECK_TEXPS_EMPTY} \\
\begin{array}{c}
\text{tnvars}^l \rightsquigarrow \text{tnvs} \\
\Delta, E_1 \vdash \text{tnvs } \overline{y_i}^i x = \text{texp} \triangleright \langle E_1^F, E_1^X \rangle \\
\overline{y_i}^i, \Delta, E \vdash \overline{td_j}^j \triangleright \langle E_2^F, E_2^X \rangle \\
\mathbf{dom}(E_1^X) \cap \mathbf{dom}(E_2^X) = \emptyset \\
\mathbf{dom}(E_1^F) \cap \mathbf{dom}(E_2^F) = \emptyset
\end{array} \\
\hline
\overline{\overline{y_i}^i, \Delta, E \vdash x l \text{tnvars}^l = \text{texp } \overline{td_j}^j \triangleright \langle E_1^F \uplus E_2^F, E_1^X \uplus E_2^X \rangle} \quad \text{CHECK_TEXPS_CONS_CONCRETE}
\end{array}$$

$\frac{\overline{y_i}^i, \Delta, E \vdash \overline{td_j}^j \triangleright \langle E^F, E^X \rangle}{\overline{y_i}^i, \Delta, E \vdash x \text{ ltnvars}^l \overline{td_j}^j \triangleright \langle E^F, E^X \rangle}$	CHECK_TEXPS_CONS_ABSTRACT
$\boxed{\delta, E \vdash id \rightsquigarrow p}$	Lookup a type class
$\frac{E(id) \triangleright p \quad \delta(p) \triangleright xs}{\delta, E \vdash id \rightsquigarrow p}$	CONVERT_CLASS_ALL
$\boxed{I \vdash (p \ t) \text{ IN } \mathcal{C}}$	Solve class constraint
$\overline{I \vdash (p \ \alpha) \text{ IN } (p_1 \text{ tnv}_1) .. (p_i \text{ tnv}_i)(p \ \alpha)(p'_1 \text{ tnv}'_1) .. (p'_j \text{ tnv}'_j)}$	SOLVE_CLASS_CONSTRAINT_IMMEDIATE
$\frac{(p_1 \text{ tnv}_1) .. (p_n \text{ tnv}_n) \Rightarrow (p \ t) \text{ IN } I \quad I \vdash (p_1 \ \sigma(\text{tnv}_1)) \text{ IN } \mathcal{C} \quad .. \quad I \vdash (p_n \ \sigma(\text{tnv}_n)) \text{ IN } \mathcal{C}}{I \vdash (p \ \sigma(t)) \text{ IN } \mathcal{C}}$	SOLVE_CLASS_CONSTRAINT_CHAIN
$\boxed{I \vdash \Sigma^{\mathcal{C}} \triangleright \mathcal{C}}$	Solve class constraints
$\frac{I \vdash (p_1 \ t_1) \text{ IN } \mathcal{C} \quad .. \quad I \vdash (p_n \ t_n) \text{ IN } \mathcal{C}}{I \vdash \{(p_1 \ t_1), .., (p_n \ t_n)\} \triangleright \mathcal{C}}$	SOLVE_CLASS_CONSTRAINTS_ALL
$\boxed{\Delta, I, E \vdash \text{val_def} \triangleright E^X}$	Check a value definition
$\frac{\Delta, E, \{\} \vdash \text{letbind} \triangleright \{\overline{x_i \mapsto t_i}^i\}, \Sigma^{\mathcal{C}}, \Sigma^{\mathcal{N}} \quad I \vdash \Sigma^{\mathcal{C}} \triangleright \mathcal{C} \quad \overline{\mathbf{FV}(t_i) \subset \text{tnvs}}^i \quad \mathbf{FV}(\mathcal{C}) \subset \text{tnvs}}{\Delta, I, E \vdash \text{let } \tau^? \text{ letbind} \triangleright \{\overline{x_i \mapsto \langle \text{forall tnv} \cdot \mathcal{C} \Rightarrow t_i, \text{let} \rangle}^i\}}$	CHECK_VAL_DEF_VAL
$\frac{\Delta, E, E^L \vdash \text{funcl}_i \triangleright \{\overline{x_i \mapsto t_i}\}, \Sigma^{\mathcal{C}}_i, \Sigma^{\mathcal{N}}_i^i \quad I \vdash \Sigma^{\mathcal{C}} \triangleright \mathcal{C} \quad \overline{\mathbf{FV}(t_i) \subset \text{tnvs}}^i \quad \mathbf{FV}(\mathcal{C}) \subset \text{tnvs} \quad \text{compatible overlap } (\overline{x_i \mapsto t_i}^i) \quad E^L = \{\overline{x_i \mapsto t_i}^i\}}{\Delta, I, E \vdash \text{let rec } \tau^? \text{ funcl}_i^i \triangleright \{\overline{x_i \mapsto \langle \text{forall tnv} \cdot \mathcal{C} \Rightarrow t_i, \text{let} \rangle}^i\}}$	CHECK_VAL_DEF_REC_FUN
$\boxed{\Delta, (\alpha_1, .., \alpha_n) \vdash t \text{ instance}}$	Check that t be a typeclass instance
$\overline{\Delta, (\alpha) \vdash \alpha \text{ instance}}$	CHECK_T_INSTANCE_VAR
$\overline{\Delta, (\alpha_1, .., \alpha_n) \vdash \alpha_1 * .. * \alpha_n \text{ instance}}$	CHECK_T_INSTANCE_TUP
$\overline{\Delta, (\alpha_1, \alpha_2) \vdash \alpha_1 \rightarrow \alpha_n \text{ instance}}$	CHECK_T_INSTANCE_FN
$\frac{\Delta(p) \triangleright \alpha'_1 .. \alpha'_n}{\Delta, (\alpha_1, .., \alpha_n) \vdash p \ \alpha_1 .. \alpha_n \text{ instance}}$	CHECK_T_INSTANCE_TC
$\boxed{\overline{z_j}^j, D_1, E_1 \vdash \text{def} \triangleright D_2, E_2}$	Check a definition

$$\begin{array}{c}
\frac{\overline{z_j}^j, \Delta_1, E \vdash \mathbf{tc} \overline{td_i}^i \triangleright \Delta_2, E^P}{\overline{z_j}^j, \Delta_1 \uplus \Delta_2, E \uplus \langle \{\}, E^P, \{\}, \{\} \rangle \vdash \overline{td_i}^i \triangleright \langle E^F, E^X \rangle} \text{CHECK_DEF_TYPE} \\
\overline{z_j}^j, \langle \Delta_1, \delta, I \rangle, E \vdash \mathbf{type} \overline{td_i}^i l \triangleright \langle \Delta_2, \{\}, \{\}, \langle \{\}, E^P, E^F, E^X \rangle \\
\\
\frac{\Delta, I, E \vdash \mathbf{val_def} \triangleright E^X}{\overline{z_j}^j, \langle \Delta, \delta, I \rangle, E \vdash \mathbf{val_def} l \triangleright \epsilon, \langle \{\}, \{\}, \{\}, E^X \rangle} \text{CHECK_DEF_VAL_DEF} \\
\\
\frac{\begin{array}{l} \Delta, E_1, E^L \vdash \mathbf{rule}_i \triangleright \{x_i \mapsto t_i\}, \Sigma^C_i, \Sigma^N_i^i \\ I \vdash \overline{\Sigma^C_i}^i \triangleright \mathcal{C} \\ \mathbf{FV}(t_i) \subset \mathbf{tnvs}^i \\ \mathbf{FV}(\mathcal{C}) \subset \mathbf{tnvs} \\ \mathbf{compatible\ overlap}(\overline{x_i \mapsto t_i}^i) \\ E^L = \{x_i \mapsto t_i^i\} \\ E_2 = \langle \{\}, \{\}, \{\}, \{x_i \mapsto \langle \mathbf{forall} \mathbf{tnvs} \mathcal{C} \Rightarrow t_i, \mathbf{let} \rangle^i\} \rangle \end{array}}{\overline{z_j}^j, \langle \Delta, \delta, I \rangle, E_1 \vdash \mathbf{indreln} \tau^? \overline{\mathbf{rule}_i}^i l \triangleright \epsilon, E_2} \text{CHECK_DEF_INDRELN} \\
\\
\frac{\overline{z_j}^j x, D_1, E_1 \vdash \mathbf{defs} \triangleright D_2, E_2}{\overline{z_j}^j, D_1, E_1 \vdash \mathbf{module} x l_1 = \mathbf{struct} \mathbf{defs} \mathbf{end} l_2 \triangleright D_2, \langle \{x \mapsto E_2\}, \{\}, \{\}, \{\} \rangle} \text{CHECK_DEF_MODULE} \\
\\
\frac{E_1(id) \triangleright E_2}{\overline{z_j}^j, D, E_1 \vdash \mathbf{module} x l_1 = id l_2 \triangleright \epsilon, \langle \{x \mapsto E_2\}, \{\}, \{\}, \{\} \rangle} \text{CHECK_DEF_MODULE_RENAME} \\
\\
\frac{\begin{array}{l} \Delta, E \vdash \mathbf{typ} \rightsquigarrow t \\ \mathbf{FV}(t) \subset \overline{\alpha_i}^i \\ \mathbf{FV}(\overline{\alpha'_k}^k) \subset \overline{\alpha_i}^i \\ \overline{\delta, E \vdash id_k \rightsquigarrow p_k}^k \\ E' = \langle \{\}, \{\}, \{\}, \{x \mapsto \langle \mathbf{forall} \overline{\alpha_i}^i. (\overline{p_k \alpha'_k})^k \Rightarrow t, \mathbf{val} \rangle^i \} \rangle \end{array}}{\overline{z_j}^j, \langle \Delta, \delta, I \rangle, E \vdash \mathbf{val} x l_1 : \mathbf{forall} \overline{\alpha_i}^i l''_i. \overline{id_k \alpha'_k l'_k}^k \Rightarrow \mathbf{typ} l_2 \triangleright \epsilon, E'} \text{CHECK_DEF_SPEC} \\
\\
\frac{\begin{array}{l} \Delta, E_1 \vdash \mathbf{typ}_i \rightsquigarrow t_i^i \\ \mathbf{FV}(t_i) \subset \overline{\alpha}^i \\ p = \overline{z_j}^j x \\ E_2 = \langle \{\}, \{x \mapsto p\}, \{\}, \{y_i \mapsto \langle \mathbf{forall} \alpha. (p \alpha) \Rightarrow t_i, \mathbf{method} \rangle^i\} \rangle \\ \delta_2 = \{p \mapsto \overline{y_i}^i\} \\ p \notin \mathbf{dom}(\delta_1) \end{array}}{\overline{z_j}^j, \langle \Delta, \delta_1, I \rangle, E_1 \vdash \mathbf{class} (x l \alpha l'') \mathbf{val} y_i l_i : \mathbf{typ}_i l_i^i \mathbf{end} l' \triangleright \langle \{\}, \delta_2, \{\}, E_2} \text{CHECK_DEF_CLASS}
\end{array}$$

$$\begin{array}{c}
E = \langle E^M, E^P, E^F, E^X \rangle \\
\Delta, E \vdash \text{typ}' \rightsquigarrow t' \\
\Delta, (\overline{\alpha_i}^i) \vdash t' \text{ \textbf{instance}} \\
tnvs = \overline{\alpha_i}^i \\
\text{duplicates}(tnvs) = \emptyset \\
\overline{\delta, E \vdash id_k \rightsquigarrow p_k}^k \\
\mathbf{FV}(\overline{\alpha'_k}^k) \subset tnvs \\
E(id) \triangleright p \\
\delta(p) \triangleright \overline{z_j}^j \\
I_2 = \{ \Rightarrow (p_k \alpha'_k)^k \} \\
\overline{\Delta, I \cup I_2, E \vdash val_def_n \triangleright E_n^X}^n \\
\text{disjoint doms}(\overline{E_n^X}^n) \\
\overline{E^X(x_k) \triangleright \langle \text{forall } \alpha''.(p \alpha'') \Rightarrow t_k, \text{method} \rangle}^k \\
\{ x_k \mapsto \langle \text{forall } tnvs. \Rightarrow \{ \alpha'' \mapsto t' \}(t_k), \text{let} \rangle^k \} = \overline{E_n^X}^n \\
\overline{x_k}^k = \overline{z_j}^j \\
I_3 = \{ (p_k \alpha'_k) \Rightarrow (p t')^k \} \\
(p \{ \alpha_i \mapsto \alpha_i''' \}(t')) \notin I
\end{array}$$

$$\overline{z_j}^j, \langle \Delta, \delta, I \rangle, E \vdash \text{instance forall } \overline{\alpha_i l'_i}^i . \overline{id_k \alpha'_k l''_k}^k \Rightarrow (id \text{ typ}') \overline{val_def_n l_n}^n \text{ \textbf{end} } l' \triangleright \langle \{ \}, \{ \}, I_3 \rangle, \epsilon$$

CHECK_DEF_

$\overline{z_j}^j, D_1, E_1 \vdash \text{defs} \triangleright D_2, E_2$

Check definitions, given module path, definitions and environment

$$\begin{array}{c}
\overline{\overline{z_j}^j, D, E \vdash \triangleright \epsilon, \epsilon} \quad \text{CHECK_DEFS_EMPTY} \\
\\
\overline{\overline{z_j}^j, D_1, E_1 \vdash \text{def} \triangleright D_2, E_2} \\
\overline{\overline{z_j}^j, D_1 \uplus D_2, E_1 \uplus E_2 \vdash \overline{def_i ; ; ?_i^i} \triangleright D_3, E_3} \\
\hline
\overline{\overline{z_j}^j, D_1, E_1 \vdash \text{def} ; ; ? \overline{def_i ; ; ?_i^i} \triangleright D_2 \uplus D_3, E_2 \uplus E_3} \quad \text{CHECK_DEFS_RELEVANT_DEF} \\
\\
\overline{E_1(id) \triangleright E_2} \\
\overline{\overline{z_j}^j, D_1, E_1 \uplus E_2 \vdash \overline{def_i ; ; ?_i^i} \triangleright D_3, E_3} \\
\hline
\overline{\overline{z_j}^j, D_1, E_1 \vdash \text{open } id \text{ l} ; ; ? \overline{def_i ; ; ?_i^i} \triangleright D_3, E_3} \quad \text{CHECK_DEFS_OPEN}
\end{array}$$

Definition rules: 145 good 0 bad
 Definition rule clauses: 437 good 0 bad