

# 1 Syntax

The minimal syntax, may extend someday.

<i>letter</i>	::=	a..z   A..Z
<i>ident</i>	::=	<i>letter</i> { <i>letter</i> }
<i>term</i>	::=	forall <i>binder</i> { <i>binder</i> }, <i>term</i>   fun { <i>binder</i> } => <i>term</i>   fix <i>ident binder</i> { <i>binder</i> } : <i>term</i> := <i>term</i>   let <i>ident</i> { <i>binder</i> } : <i>term</i> := <i>term</i> in <i>term</i>   <i>term</i> -> <i>term</i>   <i>term</i> arg { <i>arg</i> }   match <i>term</i> with       {   <i>equation</i> }   end   sort   ( <i>term</i> )
<i>arg</i>	::=	<i>term</i>
<i>binder</i>	::=	( <i>ident</i> : <i>term</i> )
<i>sort</i>	::=	Prop   Set   Type
<i>equation</i>	::=	<i>pattern</i> => <i>term</i>
<i>pattern</i>	::=	<i>ident</i> { <i>ident</i> }
<i>sentence</i>	::=	<i>axiom</i>   <i>definition</i>   <i>inductive</i>   <i>fixpoint</i>   <i>assertion proof</i>
<i>axiom</i>	::=	Axiom <i>ident</i> : <i>term</i> .
<i>definition</i>	::=	Definition <i>ident</i> { <i>binder</i> } : <i>term</i> := <i>term</i> .
<i>inductive</i>	::=	Inductive <i>ident</i> { <i>binder</i> } : <i>term</i> :=       {   <i>ident</i> : <i>term</i> } .
<i>fixpoint</i>	::=	Fixpoint <i>ident</i> { <i>binder</i> } : <i>term</i> := <i>term</i> .
<i>assertion</i>	::=	Theorem <i>ident</i> { <i>binder</i> } : <i>term</i> .
<i>proof</i>	::=	Proof . { <i>tactic</i> .} Qed .

<i>tactic</i>	::=	<i>applying</i>
		<i>context_managing</i>
		<i>case_analyzing</i>
		<i>rewriting</i>
		<i>computing</i>
		<i>equality</i>
<i>applying</i>	::=	<i>exact term</i>
		<i>apply term [in ident]</i>
<i>context_managing</i>	::=	<i>intro [ident]</i>
		<i>intros</i>
<i>case_analyzing</i>	::=	<i>destruct term</i>
		<i>induction term</i>
<i>rewriting</i>	::=	<i>rewrite [ &lt;-   -&gt; ] term [ in term ]</i>
<i>computing</i>	::=	<i>simpl</i>
<i>equality</i>	::=	<i>reflexivity</i>
		<i>symmetry</i>
 <i>helper</i>	 ::=	 <i>printing</i>
		<i>proof_handling</i>
<i>printing</i>	::=	<i>Print ident .</i>
		<i>Check term .</i>
<i>proof_handling</i>	::=	<i>Undo .</i>
		<i>Restart .</i>
		<i>Admitted .</i>
		<i>Abort .</i>

## 2 Calculus

### 2.1 Term

1.  $\text{Set}, \text{Prop}$  are terms.
2. Variables  $x, y$ , etc., are terms.
3. Constants  $c, d$ , etc., are terms.
4. If  $x$  is a variable and  $T, U$  are terms, then  $\forall x : T, U$  is a term.
5. If  $x$  is a variable and  $T, u$  are terms, then  $\lambda x : T. u$  is a term.
6. If  $x$  and  $u$  are terms, then  $(t \ u)$  is a term.
7. If  $x$  is a variable and  $t, T, u$  are terms, then  $\text{let } x := t : T \text{ in } u$  is a term.

### 2.2 Typing Rule

#### 2.2.1 Notation

- $\mathcal{S} : \{\text{Prop}, \text{Set}\}$ .
- $E$  : global environment.
- $\Gamma$  : local context.
- $u\{x/t\}$  : substitute free occurrence of variable  $x$  to term  $t$  in term  $u$ .
- $\mathcal{WF}(E)[\Gamma]$  :  $E$  is well-formed and  $\Gamma$  is valid in  $E$ .

#### 2.2.2 Typing Rules

$\mathcal{WF}([\ ])$	(T-EMPTY)
$\frac{E[\Gamma] \vdash T : s \quad s \in \mathcal{S} \quad x \notin \Gamma}{\mathcal{WF}(E)[\Gamma :: (x : T)]}$	(T-LOCAL-AX)
$\frac{E[\ ] \vdash t : T \quad c \notin E}{\mathcal{WF}(E : c := t : T)}$	(T-LOCAL-DEF)
$\frac{\mathcal{WF}(E)[\Gamma] \quad (x : T) \in \Gamma}{E[\Gamma] \vdash x : T}$	(T-VAR1)
$\frac{\mathcal{WF}(E)[\Gamma] \quad (x := t : T) \in \Gamma}{E[\Gamma] \vdash x : T}$	(T-VAR2)
$\frac{\mathcal{WF}(E)[\Gamma] \quad (c : T) \in E}{E[\Gamma] \vdash c : T}$	(T-CONST1)
$\frac{\mathcal{WF}(E)[\Gamma] \quad (c := t : T) \in E}{E[\Gamma] \vdash c : T}$	(T-CONST2)
$\frac{E[\Gamma] \vdash T : s \quad s \in \mathcal{S} \quad E[\Gamma :: (x : T)] \vdash U : \text{Prop}}{E[\Gamma] \vdash \forall x : T, U : \text{Prop}}$	(T-PROD-PROP)
$\frac{E[\Gamma] \vdash T : s \quad s \in \mathcal{S} \quad E[\Gamma :: (x : T)] \vdash U : \text{Set}}{E[\Gamma] \vdash \forall x : T, U : \text{Set}}$	(T-PROD-SET)
$\frac{E[\Gamma] \vdash \forall x : T, U : s \quad E[\Gamma :: (x : T)] \vdash t : U}{E[\Gamma] \vdash \lambda x : T. t : \forall x : T, U}$	(T-ABS)

$$\begin{array}{c}
\frac{E[\Gamma] \vdash \forall \mathbf{x} : \mathbf{U}, \mathbf{T} \quad E[\Gamma] \vdash \mathbf{u} : \mathbf{U}}{E[\Gamma] \vdash (\mathbf{t} \ \mathbf{u}) : \mathbf{T}\{\mathbf{x}/\mathbf{u}\}} \quad (\text{T-APP}) \\
\frac{E[\Gamma] \vdash \mathbf{t} : \mathbf{T} \quad E[\Gamma :: (\mathbf{x} := \mathbf{t} : \mathbf{T})] \vdash \mathbf{u} : \mathbf{U}}{E[\Gamma] \vdash \text{let } \mathbf{x} := \mathbf{t} : \mathbf{T} \text{ in } \mathbf{u} : \mathbf{U}\{\mathbf{x}/\mathbf{t}\}} \quad (\text{T-LET})
\end{array}$$

## 2.3 Conversion Rule

### 2.3.1 Notation

- $E[\Gamma] \vdash \mathbf{t} \triangleright \mathbf{u} : \mathbf{t}$  reduces to  $\mathbf{u}$  in  $E, \Gamma$  with one of the  $\beta, \iota, \delta, \zeta$  reductions.
- $E[\Gamma] \vdash \mathbf{t} \triangleright^* \mathbf{u} : E[\Gamma] \vdash \mathbf{t} \triangleright \dots \triangleright \mathbf{u}$ .
- $\mathbf{u} \equiv \mathbf{v} : \mathbf{u}$  and  $\mathbf{v}$  are identical.

### 2.3.2 Conversion Rules

$$\begin{array}{c}
\frac{E[\Gamma] \vdash (\lambda \mathbf{x} : \mathbf{T}. \mathbf{t}) \ \mathbf{u}}{\mathbf{t}\{\mathbf{x}/\mathbf{u}\}} \quad (\beta\text{-CONV}) \\
\frac{\text{case}((\mathbf{c}_p \ \mathbf{q}_1 \ \dots \ \mathbf{q}_r \ \mathbf{a}_1 \ \dots \ \mathbf{a}_m), \mathbf{P}, \mathbf{f}_1 | \dots | \mathbf{f}_n)}{f_i \ \mathbf{a}_1 \ \dots \ \mathbf{a}_m} \quad (\iota\text{-CONV}) \\
\frac{E[\Gamma] \vdash \mathbf{x} \quad (\mathbf{x} := \mathbf{t} : \mathbf{T}) \in \Gamma}{\mathbf{t}} \quad (\delta\text{-CONV1}) \\
\frac{E[\Gamma] \vdash \mathbf{c} \quad (\mathbf{x} := \mathbf{t} : \mathbf{T}) \in E}{\mathbf{t}} \quad (\delta\text{-CONV2}) \\
\frac{E[\Gamma] \vdash \text{let } \mathbf{x} := \mathbf{u} \text{ in } \mathbf{t}}{\mathbf{t}\{\mathbf{x}/\mathbf{u}\}} \quad (\zeta\text{-CONV}) \\
\frac{E[\Gamma] \vdash \mathbf{t} : \forall \mathbf{x} : \mathbf{T}, \mathbf{U} \quad \mathbf{x} \text{ fresh in } \mathbf{t}}{\lambda \mathbf{x} : \mathbf{T}. (\mathbf{t} \ \mathbf{x})} \quad (\eta\text{-EXP})
\end{array}$$

**Definition 1** (Convertibility).  $\mathbf{t}_1$  and  $\mathbf{t}_2$  are convertible iff there exists  $\mathbf{u}_1$  and  $\mathbf{u}_2$  such that  $E[\Gamma] \vdash \mathbf{t}_1 \triangleright^* \mathbf{u}_1$  and  $E[\Gamma] \vdash \mathbf{t}_2 \triangleright^* \mathbf{u}_2$  and either  $\mathbf{u}_1 \equiv \mathbf{u}_2$  or they are convertible up to  $\eta$ -expansion.

## 2.4 Inductive Definition

### 2.4.1 Notation

- $\text{Ind}[p](\Gamma_I := \Gamma_C) : \text{inductive definition.}$
- $\Gamma_I : \text{names and types of inductive type.}$
- $\Gamma_C : \text{names and types of constructors of inductive type.}$
- $p : \text{the number of parameters of inductive type.}$
- $\Gamma_P : \text{the context of parameters.}$

### 2.4.2 Typing Rule

$$\begin{array}{c}
\frac{\mathcal{WF}(E)[\Gamma] \quad \text{Ind}[p](\Gamma_I := \Gamma_C) \in E \quad (\mathbf{a} : \mathbf{A}) \in \Gamma_i}{E[\Gamma] \vdash \mathbf{a} : \mathbf{A}} \quad (\text{T-IND}) \\
\frac{\mathcal{WF}(E)[\Gamma] \quad \text{Ind}[p](\Gamma_I := \Gamma_C) \in E \quad (\mathbf{c} : \mathbf{C}) \in \Gamma_C}{E[\Gamma] \vdash \mathbf{c} : \mathbf{C}} \quad (\text{T-CONSTR}) \\
\frac{(E[\Gamma_P] \vdash \mathbf{A}_j : \mathbf{s}'_j)_{j=1..k} \quad (E[\Gamma_i; \Gamma_P] \vdash \mathbf{C}_i : \mathbf{s}_{\mathbf{q}_i})_{i=1..n}}{\mathcal{WF}(E; \text{Ind}[p](\Gamma_I := \Gamma_C))[\Gamma]} \quad (\text{T-WF-IND})
\end{array}$$

### 2.4.3 Well-formed Requirement

To maintain the consistency of the system, we must restrict the inductive definitions to a syntactic criterion of **positivity**, which guarantees the *soundness and safety* of the system.

**Definition 2** (Constructor). *T is a type of constructor of I if*

- $T \equiv (I \ t_1 \ \cdots \ t_n)$
- $T \equiv \forall x : U, T'$ , where  $T'$  is a type of constructor of I

**Definition 3** (Positivity). *The type of constructor T satisfies the positivity condition for a constant X if*

- $T \equiv (X \ t_1 \ \cdots \ t_n)$  and X does not occur free in  $t_i$
- $T \equiv \forall x : U, V$  and X occurs only *strictly positively* in U and V satisfies the positivity condition for X

**Definition 4** (Strictly Positivity). *The constant X occurs strictly positively in T if*

- X does not occur in T
- $T \triangleright^* (X \ t_1 \ \cdots \ t_n)$  and X does not occur in  $t_i$
- $T \triangleright^* \forall x : U, V$  and X does not occur in U but occurs *strictly positively* in V
- $T \triangleright^* (I \ a_1 \ \cdots \ a_m \ t_1 \ \cdots \ t_p)$ , where  $\text{Ind}[m](I : A := c_1 : \forall p_1 : P_1, \dots, \forall p_m : P_m, C_1; \dots; c_n : \forall p_1 : P_1, \dots, \forall p_m : P_m, C_n)$ , and X does not occur in  $t_i$ , and the types of constructor  $C_i\{p_j/a_j\}_{j=1..m}$  satisfies the nested positivity condition for X

**Definition 5** (Nested Positivity). *The type of constructor T satisfies the nested positivity condition for a constant X if*

- $T \equiv (I \ b_1 \ \cdots \ b_m \ u_1 \ \cdots \ u_p)$ , where I is an inductive definition with m parameters and X does not occur in  $u_i$
- $T \equiv \forall x : U, V$  and X occurs *strictly positively* in U and V satisfies the nested positivity condition for X

## 3 Destructor

## 4 Fixpoint

## References

- [1] The Coq Development Team. *The Coq Proof Assistant Reference Manual*, 8.7.2 edition, February 2018.