All Coq Rules in One Place

Xiaohong Chen Lucas Peña

May 25, 2020

Abstract

This document summarizes all the proof rules of the Coq proof assistant, as listed in `https://coq.inria.fr/distrib/current/refman/language/cic.html`.

1 Syntax

Let us fix a countably infinite set V of *variables*, denoted $x, y, \ldots$. Let us fix a countably infinite set C of *constants*, denoted $c, d, \ldots$.

Definition 1.1. We define the set *Term* to be the smallest set that satisfies the following conditions:

1. $\mathsf{SProp}, \mathsf{Prop}, \mathsf{Set} \in \mathit{Term}$; $\mathsf{Type}(i) \in \mathit{Term}$ for every $i \in \mathbb{N}_{\geq 1}$.
2. $V \subseteq \mathit{Term}$.
3. $C \subseteq \mathit{Term}$.
4. If $x \in V$ and $T, U \in \mathit{Term}$, then $\forall x\!:\!T, U \in \mathit{Term}$.
5. If $x \in V$ and $T, u \in \mathit{Term}$, then $\lambda x\!:\!T.u \in \mathit{Term}$.
6. If $t, u \in \mathit{Term}$, then $(t\, u) \in \mathit{Term}$, called *application*.
7. If $x \in V$ and $t, T, u \in \mathit{Term}$, then $\mathsf{let}\, x \coloneqq t\!:\!T \,\mathsf{in}\, u \in \mathit{Term}$.

where $\forall x\!:\!T, U$ binds x to U and $\lambda x\!:\!T.u$ binds x to u. We use $\mathrm{FV}(T) \subseteq V$ to denote the set of free variables in $T \in \mathit{Term}$. For $T, U \in \mathit{Term}$ and $x \in V$, we use $T[U/x]$ to denote the result of substituting U for x in T, where α-renaming happens implicitly to prevent variable capture.

Definition 1.2. We define the set $\mathit{Sort} = \{\mathsf{SProp}, \mathsf{Prop}, \mathsf{Set}\} \cup \{\mathsf{Type}(i) \mid i \in \mathbb{N}\}$. Note that $\mathit{Sort} \subseteq \mathit{Term}$. Elements in *Sort* are called *sorts* and denoted as s, possibly with subscripts.

Definition 1.3. A *local assumption* is written $x\!:\!T$, where $x \in V$ and $T \in \mathit{Term}$. A *local definition* is written $x \coloneqq u\!:\!T$, where $x \in V$ and $u, T \in \mathit{Term}$. In both cases, we call x the *declared variable*. A *local context* is an ordered list of local assumptions and local definitions, such that the declared variables are all distinct. We use Γ, possibly with subscripts, to denote local contexts.

Notation 1.4. We use the notation $[x\!:\!T \,;\, y \coloneqq u\!:\!U \,;\, z\!:\!V]$ to denote the local context that consists of the local assumption $x\!:\!T$, the local definition $y \coloneqq u\!:\!U$ and the local assumption $z\!:\!V$, with the implicit requirement that x, y, z are all distinct. The empty local context is written as $[]$. Let Γ be a local context. We write $x \in \Gamma$ to mean that x is declared in Γ. We write $(x\!:\!T) \in \Gamma$ to mean that the local assumption $x\!:\!T$ is in Γ, or that the local definition $x \coloneqq u\!:\!T$ is in Γ for some $u \in \mathit{Term}$. We write $(x \coloneqq u\!:\!T) \in \Gamma$ to mean that the local definition $x \coloneqq u\!:\!T$ is in Γ. We write $\Gamma :: (x\!:\!T)$ to denote the local context that enriches Γ with $x\!:\!T$, with the implicit requirement that $x \notin \Gamma$. Similarly, we write $\Gamma :: (x \coloneqq u\!:\!T)$ to denote the local context that enriches Γ with $x \coloneqq u\!:\!T$, with the implicit requirement that $x \notin \Gamma$. We write $\Gamma_1 \,;\, \Gamma_2$ to mean the local context obtained by concatenating Γ_1 and Γ_2, with the implicit requirement that all variables declared in Γ_2 are not declared in Γ_1.

Definition 1.5. A *global assumption* is written $(c : T)$, with the parentheses, where $c \in C$ and $T \in \mathit{Term}$. A *global definition* is written $c \coloneqq u : T$, where $c \in C$ and $u, T \in \mathit{Term}$. In both cases, we call c the *declared constant*. A *global environment* is an ordered list of global assumptions and global definitions, and also *declarations of inductive objects*, which are defined later. We use E, possibly with subscripts, to denote global environments.

Notation 1.6. We use the notation $c_1 : T \,;\, c_2 \coloneqq u : U \,;\, c_3 : V$ to denote the local context that consists of the global assumption $c_1 : T$, the global definition $c_2 \coloneqq u : U$ and the global assumption $c_3 : V$. The empty global context is written as $[]$. Let E be a local context. We write $c \in E$ to mean that c is declared in E. We write $(c : T) \in E$ to mean that the global assumption $c : T$ is in E, or that the global definition $c \coloneqq u : T$ is in E for some $u \in \mathit{Term}$. We write $(c \coloneqq u : T) \in E$ to mean that the global definition $c \coloneqq u : T$ is in E. We write $E \,;\, c : T$ to denote the global context that enriches E with $c : T$. Similarly, we write $E \,;\, c \coloneqq u : T$ to denote the global context that enriches E with $(c \coloneqq u : T)$.

Notation 1.7. We write $E[\Gamma] \vdash u : T$ to mean that u is *well-typed* with type T in global environment E and local environment Γ. We write $\mathcal{WF}(E)[\Gamma]$ to mean that the global environment E is *well-formed* and Γ is a *valid local context* in E.

Definition 1.8. A term u is *well-typed* in a global environment E if there is a local context Γ and type T such that $E[\Gamma] \vdash u : T$ is derivable with the rules below.

2 Coq Rules

In this section we list all Coq rules. A *rule* consists of a set of *premises* and one *conclusion*, separated by a horizontal bar. For readability, we put *side conditions* alongside the premises. Side conditions are typed in green, to distinguish from the premises.

2.1 Basic Typing Rules

There are 18 basic typing rules, as shown below.

Names	**Rules**	**Comments**
(W-Empty)	$\dfrac{\cdot}{\mathcal{WF}([])[]}$	The empty global environment is well-formed, and the empty local context is a valid local context in the empty global environment.
(W-Local-Assum)	$\dfrac{E[\Gamma] \vdash T : s \quad s \in S \quad x \notin \Gamma}{\mathcal{WF}(E)[\Gamma :: (x : T)]}$	The side condition $x \notin \Gamma$ needs not to be specified because it is implicit in the notation $\Gamma :: (x : T)$; see Notation 1.4.

(W-Local-Def) $$\frac{E[\Gamma] \vdash t : T \quad x \notin \Gamma}{\mathcal{WF}(E)[\Gamma :: (x \coloneqq t : T)]}$$

The side condition $x \notin \Gamma$ needs not to be specified because it is implicitly implied by the notation $\Gamma :: (x \coloneqq t : T)$; see Notation 1.4.

(W-Global-Assum) $$\frac{E[] \vdash T : s \quad s \in S \quad c \notin E}{\mathcal{WF}(E; c : T)[]}$$

(W-Global-Def) $$\frac{E[] \vdash t : T \quad c \notin E}{\mathcal{WF}(E; c \coloneqq t : T)[]}$$

(Ax-SProp) $$\frac{\mathcal{WF}(E)[\Gamma]}{E[\Gamma] \vdash \mathsf{SProp} : \mathsf{Type}(1)}$$

(Ax-Prop) $$\frac{\mathcal{WF}(E)[\Gamma]}{E[\Gamma] \vdash \mathsf{Prop} : \mathsf{Type}(1)}$$

(Ax-Set) $$\frac{\mathcal{WF}(E)[\Gamma]}{E[\Gamma] \vdash \mathsf{Set} : \mathsf{Type}(1)}$$

(Ax-Type) $$\frac{\mathcal{WF}(E)[\Gamma]}{E[\Gamma] \vdash \mathsf{Type}(i) : \mathsf{Type}(i+1)}$$

Here, $i \in \mathbb{N}_{\geq 1}$ is any positive natural number.

(Var) $$\frac{\mathcal{WF}(E)[\Gamma] \quad (x : T \in \Gamma), \text{ or } (x \coloneqq t : T) \in \Gamma \text{ for some } t \in \mathit{Term}}{E[\Gamma] \vdash x : T}$$

(Const) $$\frac{\mathcal{WF}(E)[\Gamma] \quad (c : T \in E), \text{ or } (c \coloneqq t : T) \in E \text{ for some } t \in \mathit{Term}}{E[\Gamma] \vdash c : T}$$

(Prod-SProp) $$\frac{E[\Gamma] \vdash T : s \quad s \in S \quad E[\Gamma :: (x : T)] \vdash U : \mathsf{SProp}}{E[\Gamma] \vdash \forall x : T, U : \mathsf{SProp}}$$

(Prod-Prop) $$\frac{E[\Gamma] \vdash T : s \quad s \in S \quad E[\Gamma :: (x : T)] \vdash U : \mathsf{Prop}}{E[\Gamma] \vdash \forall x : T, U : \mathsf{Prop}}$$

(Prod-Set) $$\frac{E[\Gamma] \vdash T : s \quad s \in \{\mathsf{SProp}, \mathsf{Prop}, \mathsf{Set}\} \quad E[\Gamma :: (x : T)] \vdash U : \mathsf{Set}}{E[\Gamma] \vdash \forall x : T, U : \mathsf{Set}}$$

(Prod-Type) $$\frac{E[\Gamma] \vdash T : s \quad s \in \{\mathsf{SProp}, \mathsf{Type}(i)\} \quad E[\Gamma :: (x : T)] \vdash U : \mathsf{Type}(i)}{E[\Gamma] \vdash \forall x : T, U : \mathsf{Type}(i)}$$

(Lam) $$\frac{E[\Gamma] \vdash \forall x : T, U : s \quad s \in S \quad E[\Gamma :: (x : T)] \vdash t : U}{E[\Gamma] \vdash \lambda x : T.t : \forall x : T, U}$$

(App) $$\frac{E[\Gamma] \vdash t : \forall x : U, T \quad E[\Gamma] \vdash u : U}{E[\Gamma] \vdash (t\, u) : T[u/x]}$$

(Let) $$\frac{E[\Gamma] \vdash t : T \quad E[\Gamma :: (x \coloneqq t : T)] \vdash u : U}{E[\Gamma] \vdash \mathsf{let}\, x \coloneqq t : T \,\mathsf{in}\, u : U[t/x]}$$

2.2 Conversion Rules

In this section, we define what it means for two Coq programs to be *intentionally equal*, or *convertible*.

Names	**Rules**	**Comments**
(Beta)	$\frac{}{E[\Gamma] \vdash ((\lambda x : T.t)u) \triangleright_\beta t[x/u]}$	We say that $t[x/u]$ is the β-*contraction* of $((\lambda x : T.t)u)$, and that $((\lambda x : T.t)u)$ is the β-*expansion* of $t[x/u]$
		ι-reduction rules to be defined later
(Delta-Local)	$\frac{\mathcal{WF}(E)[\Gamma] \quad (x \coloneqq t : T) \in \Gamma}{E[\Gamma] \vdash x \triangleright_\Delta t}$	Reducing variable defined in local context

$$\text{(Delta-Local)} \quad \frac{\mathcal{WF}(E)[\Gamma] \quad (c \coloneqq t : T) \in E}{E[\Gamma] \vdash c \triangleright_\delta t}$$

Reducing constant defined in global context

$$\text{(Zeta)} \quad \frac{\mathcal{WF}(E)[\Gamma] \quad E[\Gamma] \vdash u : U \quad E[\Gamma :: (x \coloneqq u : U)] \vdash t : T}{E[\Gamma] \vdash \mathsf{let}\, x \coloneqq u : U \,\mathsf{in}\, t \triangleright_\zeta t[x/x]}$$

Remove local definitions occurring in terms

In addition to the above convertibility rules, we also allow identifying a term t of type $\forall x : t, U$ with its *η-expansion* $\lambda x : T.(tx)$ for x fresh in t. Note *η-reduction* is deliberately not defined.

Notation 2.1. We write $E[\Gamma] \vdash t \triangleright u$ for the contextual closure of the rules defined above. That is, t reduces to u with global environment E and local context Γ with one of the previous reductions $\beta, \Delta, \delta, \iota$, or ζ.

Definition 2.2. Two terms are called *irrelevant* if they share a common type of a strict proposition $A : \mathsf{SProp}$. Irrelevant terms can be identified.

Definition 2.3. Two terms t_1, t_2 are called *$\beta\delta\iota\zeta\eta$-convertible*, or *convertible*, or *equivalent* in global environment E and local context Γ iff there exists t_1, t_2 such that

$$E[\Gamma] \vdash t_1 \triangleright \ldots \triangleright u_1 \text{ and } E[\Gamma] \vdash t_2 \triangleright \ldots \triangleright u_2$$

and either u_1 and u_2 are identical up irrelevant subterms, or they are convertible up to η-exxpansion. We denote this as $E[\Gamma] \vdash t_1 =_{\beta\delta\iota\zeta\eta} t_2$

2.3 Subtyping Rules

The *subtyping* relation is inductively defined as follows:

- if $E[\Gamma] \vdash t =_{\beta\delta\iota\zeta\eta} u$, then $E[\Gamma] \vdash t \leq_{\beta\delta\iota\zeta\eta} u$
- if $i \leq j$, then $E[\Gamma] \vdash \mathsf{Type}(i) \leq_{\beta\delta\iota\zeta\eta} \mathsf{Type}(j)$
- $E[\Gamma] \vdash \mathsf{Set} \leq_{\beta\delta\iota\zeta\eta} \mathsf{Type}(i)$ for all i
- $E[\Gamma] \vdash \mathsf{Prop} \leq_{\beta\delta\iota\zeta\eta} \mathsf{Set}$
- if $E[\Gamma] \vdash T =_{\beta\delta\iota\zeta\eta} U$ and $E[\Gamma :: (x : T)] \vdash T' \leq_{\beta\delta\iota\zeta\eta} U'$, then $E[\Gamma] \vdash \forall x : T, T' \leq_{\beta\delta\iota\zeta\eta} \forall x : U, U'$

2.4 Conversion/Subtyping: Polymorphic Universes

2.5 Conversion Typing Rule

We now have the infrastructure to define the typing rule for conversion:

$$\text{(Conv)} \quad \frac{E[\Gamma] \vdash U : s \quad E[\Gamma] \vdash t : T \quad E[\Gamma] T \leq_{\beta\delta\iota\zeta\eta} U}{E[\Gamma] \vdash t : U}$$

2.6 Inductive Definitions

Definition 2.4. We represent an *inductive definition* as $\mathsf{Ind}[p]\,(\Gamma_I := \Gamma_C)$ where

- p represents the number of parameters of the inductive types
- Γ_I represents the names and types of inductive types
- Γ_C represents the names and types of the constructors of the inductive types

Definition 2.5. Let $\mathsf{Ind}[p]\,(\Gamma_I := \Gamma_C)$ be an inductive definition and let T be such that $(t : T) \in \Gamma_I \cup \Gamma_C$. Then, there exists a $\Gamma_P = [a_1 : A_1\,; \ldots ; a_p : A_p]$ such that T can be written as $\forall \Gamma_P, T'$ for some T'. Here Γ_P is called the *context of paramters*.

Additionaly, if $(t : T) \in \Gamma_I$, then T can be written as $\forall \Gamma_P, \Gamma_{Arr(t)}, s$. Here, $Arr(t)$ is called the *Arity* of the inductive type t and s is called the sort of t.

Example 2.6. The inductive definition for parameterized lists is:

$$\mathsf{Ind}[1]\left(\left[\mathsf{list} : \mathsf{Set} \to \mathsf{Set}\right] := \begin{bmatrix} \mathsf{nil} & : \forall A : \mathsf{Set},\ \mathsf{list}\,A \\ \mathsf{cons} & : \forall A : \mathsf{Set},\ A \to \mathsf{list}\,A \to \mathsf{list}\,A \end{bmatrix}\right)$$

This corresponds to the Coq definition:

```
Inductive list (A:Set) : Set :=
| nil : list A
| cons : A -> list A -> list A.
```

Below are rules for types of constants in a global environment that contain an inductive definition:

$$(\textsc{Ind})\quad \frac{\mathcal{WF}(E)[\Gamma] \quad \mathsf{Ind}[p]\,(\Gamma_I := \Gamma_C) \in E \quad a : A \in \Gamma_I}{E[\Gamma] \vdash a : A}$$

$$(\textsc{Constr})\quad \frac{\mathcal{WF}(E)[\Gamma] \quad \mathsf{Ind}[p]\,(\Gamma_I := \Gamma_C) \in E \quad c : C \in \Gamma_C}{E[\Gamma] \vdash c : C}$$

Definition 2.7. A type T is an *arity of sort s* if it converts to the sort s or to a product $\forall x : T, U$ with U an arity of sort s.

Definition 2.8. A type T is an *arity* if there is an $s \in S$ such that T is an arity of sort s.

Definition 2.9. A type T is a *type of constructor of I* if T is $(I\,t_1 \ldots t_n)$ or T is $\forall x : U, T'$ where T' is a type of constructor of I.

Definition 2.10. A type of constructor T is said to *satisfy the positivity condition* for X if $T = (X\,t_1 \ldots t_n)$ and X does not occur free in any t_i, or $T = \forall x : U, V$ where X occurs only strictly positively in U and V satisfies the positivity condition for X.

Definition 2.11. A constant X occurs *strictly positive* in T if one of the following cases hold:

- X does not occur in T
- T converts to $(X\,t_1 \ldots t_n)$ and X does not occur in any T_i
- T converts to $\forall x : U, V$ where X does not occur in U and X occurs strictly positively in V
- T converts to $(I\,a_1 \ldots a_m\,t_1 \ldots t_p)$ where I represents the inductive definition

  $$\mathsf{Ind}[m]\,(I : A := c_1 : \forall p_1 : P_1, \ldots \forall p_m : P_m, C_1\,; \ldots c_n : \forall p_1 : P_1, \ldots \forall p_m : P_m, C_n)$$

  and X does not occur in any of the t_i, and all instantiated types of constructors $C_i\{p_j/a_j\}_{j=1\ldots m}$ satisfy the nested positivity condition for X

Definition 2.12. A type of constructor T of I is said to *satisfy the nested positivity condition* for X if $T = (I\,b_1 \ldots b_n u_1 \ldots u_p)$ where I is an inductive type with m paramters and X does not occur in any u_i, or $T = \forall x : U, V$ where X occurs only strictly positively in U and V satisfies the nested positivity condition for X.

2.6.1 Correctness Rules

Let E be a global environment and $\Gamma_P, \Gamma_I, \Gamma_C$ be contexts such that Γ_I is $[I_1 : \forall \Gamma_P, A_1 ; \ldots I_k : \forall \Gamma_P, A_k]$ and Γ_C is $[c_1 : \forall \Gamma_P, C_1 ; \ldots c_n : \forall \Gamma_P, C_n]$. Then we have the following well-formedness rule:

$$\text{(Ind)} \qquad \frac{\mathcal{WF}(E)[\Gamma_P] \quad (E[\Gamma_I ; \Gamma_P] \vdash C_i : s_{q_i})_i = 1 \ldots n}{\mathcal{WF}(E ; \mathsf{Ind}[p]\,(\Gamma_I \coloneqq \Gamma_C))[]}$$

provided the following side conditions hold:

- $k > 0$ and all I_j and c_i are distinct names for $j = 1 \ldots k$ and $i = 1 \ldots n$
- p is the number of parameters of $\mathsf{Ind}[p]\,(\Gamma_I \coloneqq \Gamma_C)$ and Γ_P is the context of paramters
- for $j = 1 \ldots k$, A_j is an arity of sort s_j and $I_j \notin E$
- for $i = 1 \ldots n$, C_i is a type of constructor of I_{q_i} which satisfies the positivity condition for $I_1 \ldots I_k$ and $c_i \notin E$