

Instruction For The Use Of (.v)Files

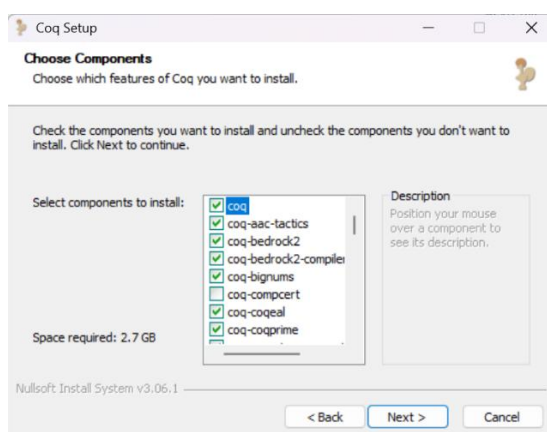
Coq is an open-source formal proof management system. It provides a formal language to write mathematical definitions, executable algorithms and theorems together with an environment for semi-interactive development of machine-checked proofs. [\[Coq website\]](#)

This instruction aims to introduce the most basic usage for running code in CoqIDE, helping reviewers and readers to verify the integrity and correctness of our Coq code. For other more details of the usage of Coq, see <https://rocq-prover.org/docs>.

1. Install the CoqIDE (version 8.20.1).

The formalization is implemented in the CoqIDE (version 8.20.1) for 64 bit Windows. Here is the [download link](#).

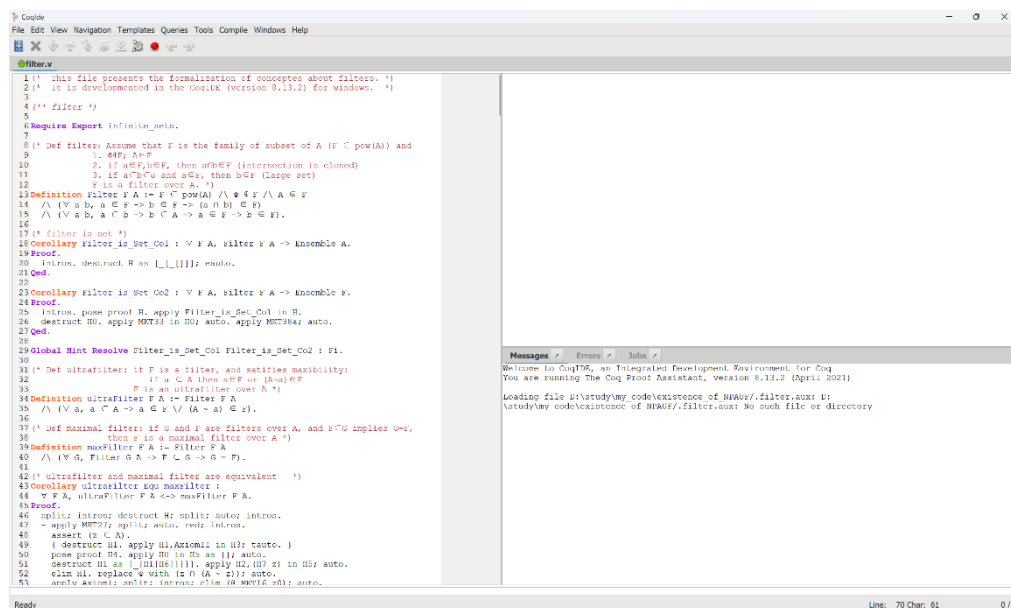
Open the downloaded installer and proceed with the installation.



Try to ensure that the installation path is in English. After installation, (.v)files can be opened and edited by CoqIDE.

2. Interface of CoqIDE.

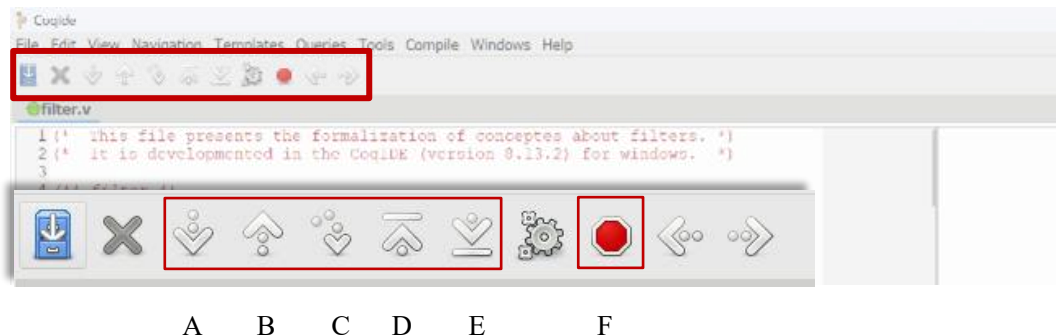
The CoqIDE interface consists mainly of three windows:



The left area is for code writing; the top right window is for interaction, where you can observe the validation process of proofs while running code, and then continue writing and modifying proof code based on this feedback; the bottom right window is for information display, which can show error messages and print definitions and theorem contents that have been formalized.

3. The execution of code.

Coq code consists of various internal commands, which can be executed by clicking on the buttons indicated in the figure below.



A: forward one command (ctrl + down)

B: backward one command (ctrl + up)

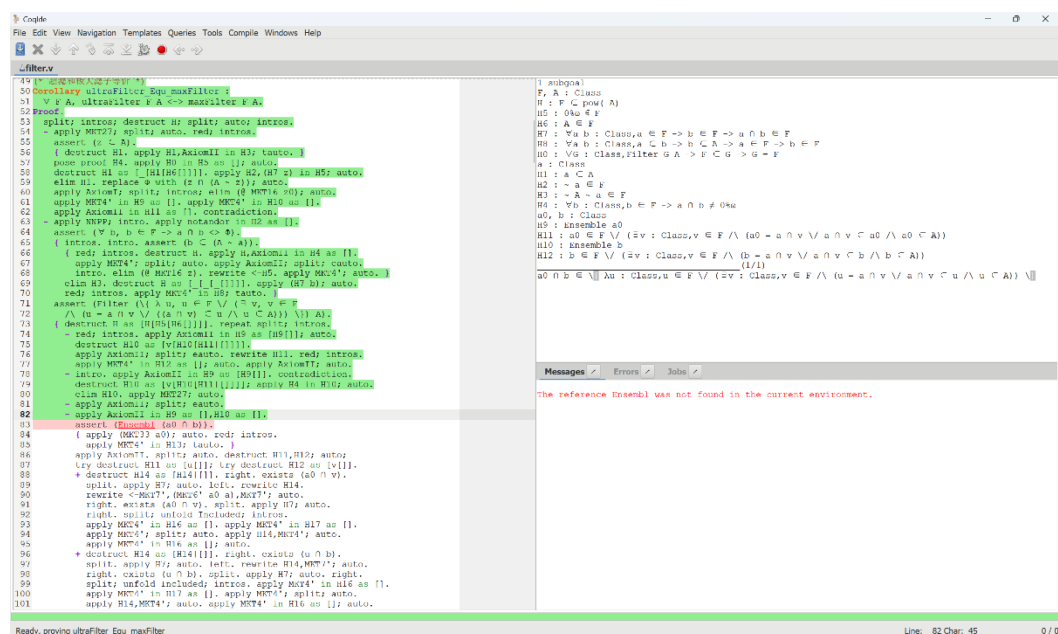
C: run to cursor (ctrl + right)

D: restart Coq

E: run to the end

F: interrupt the execution

The code executed successfully will be displayed in green, and the proof process will be presented in the interactive area. If some command cannot be executed, it will be displayed in red and the error message be displayed in the bottom right window.



Especially, some code can be executed successfully but will be displayed in yellow. This is divided into two cases: one is that the command “**admit**” or “**Admitted**” is used:

```

16
17 (* filter is set *)
18 Corollary Filter_is_Set_Col :  $\forall F A, \text{Filter } F A \rightarrow \text{Ensemble } A.$ 
19 Proof.
20   intros. destruct H as [_[[]]]. admit.
21 Admitted.
22
23 Corollary Filter_is_Set_Co2 :  $\forall F A, \text{Filter } F A \rightarrow \text{Ensemble } F.$ 
24 Proof.
25   intros. pose proof H. apply Filter_is_Set_Col in H.
26   destruct H0. apply MKT33 in H0; auto.
27 Admitted.
28

```

the other is that the code is declared by the command “**Parameter**” or “**Axiom**”:

```

--
313 Parameter MK_Axiom : MK_Axioms.
314
315 Notation AxiomI := (@ A I MK_Axiom).
316 Notation AxiomII := (@ A_II MK_Axiom).
317 Notation AxiomIII := (@ A_III MK_Axiom).
318 Notation AxiomIV := (@ A_IV MK_Axiom).
319 Notation AxiomV := (@ A_V MK_Axiom).
320 Notation AxiomVI := (@ A_VI MK_Axiom).
321 Notation AxiomVII := (@ A_VII MK_Axiom).
322 Notation AxiomVIII := (@ A_VIII MK_Axiom).
323
324 Axiom AxiomIX :  $\exists c, \text{ChoiceFunction } c \wedge \text{dom}(c) = \mu \sim [\Phi].$ 
325

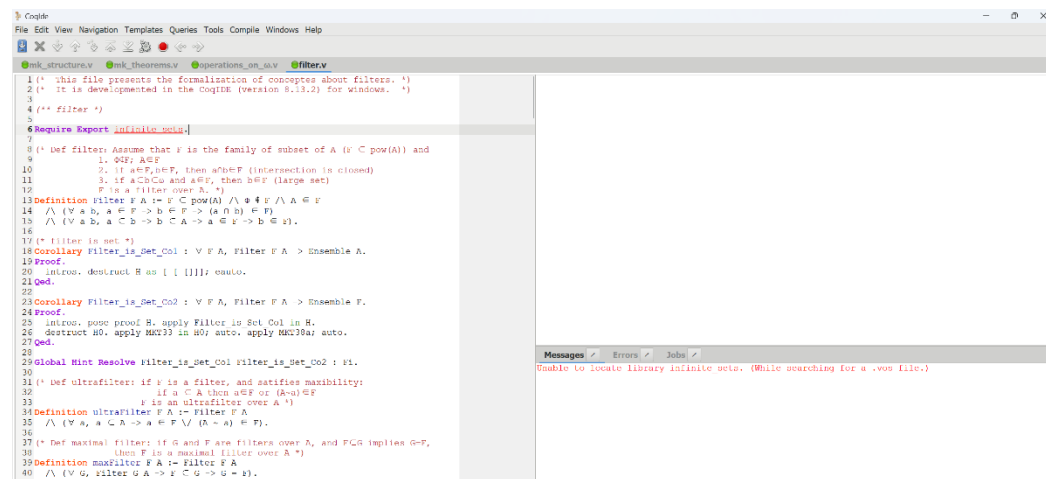
```

For the first case, it is not permitted because the command “**admit**” and “**Admitted**” indicate that the proof is not finished, and the display in yellow is to remind developers of completing it.

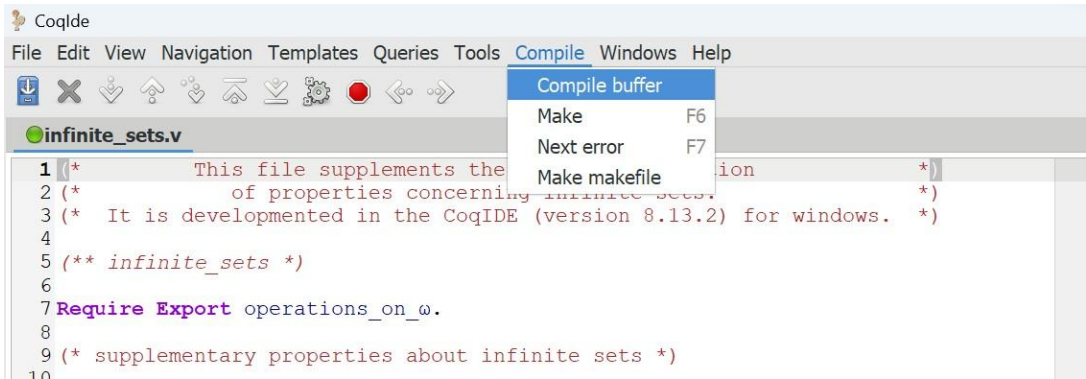
The second case is conditionally permitted. The command “**Parameter**” and “**Axiom**” are utilized to describe propositions that are temporarily unprovable or objects that are temporarily unconstructible, such as the description of axioms, which is common in mathematics. Therefore, these two commands are not to be used casually; they are typically reserved for describing widely recognized or rigorously proven consistent axioms and other objects.

4. The compilation of code.

In general, a (.v)file may depend on other (.v)files (usually read with the “**Export**” or “**Import**” command). In this case, the dependent files need to be compiled first, otherwise, it cannot run. For example, in the figure below, the dependent file is not compiled, errors occur when running.

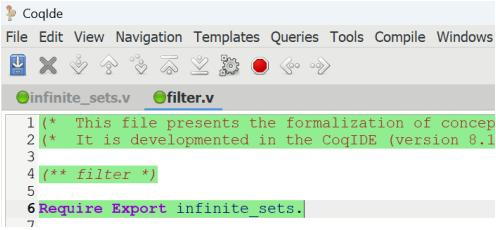


To compile the dependent file, click on “Compile” and “Compile buffer” in the interface of the file to be compiled as shown in the figure below.



Make sure that the code to be compiled can be successfully executed in green or yellow, otherwise it cannot be compiled. If compilation succeeds, a corresponding (.vo)file will be generated of the same name and path as the compiled (.v)file. After this, the dependent file can be read by other files:

infinite_sets.v
infinite_sets.vo



Note that the dependent file and the other files that read it should be in the same path.

Run all files according to the instructions above, until all the code runs successfully and turns green (or yellow, which should be carefully checked). This indicates that all formal definitions and theorems are correctly described and verified.

This document was personally edited by Dou(dgw@bupt.edu.cn). Please do not distribute it without permission.