

HLF 2015, Aug. 25, 2015, Heidelberg.

# UniMath

by Vladimir Voevodsky  
from the Institute for Advanced Study in Princeton, NJ.

**Definition.** UniMath is a library of formalized mathematics that the UniMath development team is developing.

It is freely available and can be found on GitHub. Since recently it uses of-the-shelf version of Coq. I refer to the documentation available there for the mundane information about the library.

Today I want to talk about the principles that guide the choices that we have to make while working on the UniMath.

There are several principles that the development team has decided to follow:

1. UniMath is written based on the univalent approach to formalization of mathematics.
2. UniMath is written in the language of the proof assistant Coq.
3. UniMath uses a subset of features of the proof assistant that have been verified to be consistent relative to the set theory.
4. The features that UniMath uses are consistent not only by themselves but also when combined with the ***Axiom of Excluded Middle*** (for propositions).

The fourth principle, the condition that the set of features that UniMath uses remains consistent if one adds the axiom of excluded middle, means that everything proved in UniMath can be used in projects that aim to formalize classical mathematics.

It is however the third principle that is most characteristic of UniMath and that is the most difficult one to satisfy in practice.

Today we can only claim that we work hard on having it satisfied and that it carries a very high weight in the decision process that leads to various practical choices that we make for the library.

Let me try to explain in what sense we would like to see the word “verified” to be understood here.

Let us say that a set of features is considered to be *informally* verified if

1. It has been fully described and the description has been published.
2. At least one proof checking algorithm for proofs based on this set of features have been fully described and the description has been published.
3. At least two independent open source implementations of the published proof checking algorithm are available.
4. A univalent interpretation for a subset of the language that contains all of the sentences that are approved by the published proof checking algorithm have been fully described and published.

These four conditions, in my opinion, form the minimal set of requirements and only allow to call a given set of features to be *informally* verified. For a set of features to be *semi-formally* verified one should add to this set the fifth condition:

5. The published description of the univalent interpretation used in the fourth condition have been formalized in an independently trusted proof assistant such as, for example, HOL Light.

For the set of features to be **formally** verified we should add the sixth and the seventh conditions:

6. The published proof checking algorithm have been formalized and formally verified to satisfy the fourth condition using an independently trusted formal verification system.

7. At least one implementation of the published proof checking algorithm have been formally verified in an independently trusted formal verification system.

Of the four conditions required to achieve the informal level of consistency verification the first three belong purely to the realm of computer science. They concern the description and implementation of what is, essentially, a data structure and an algorithm for manipulating elements of this data structure.

The fourth condition is of a different nature.

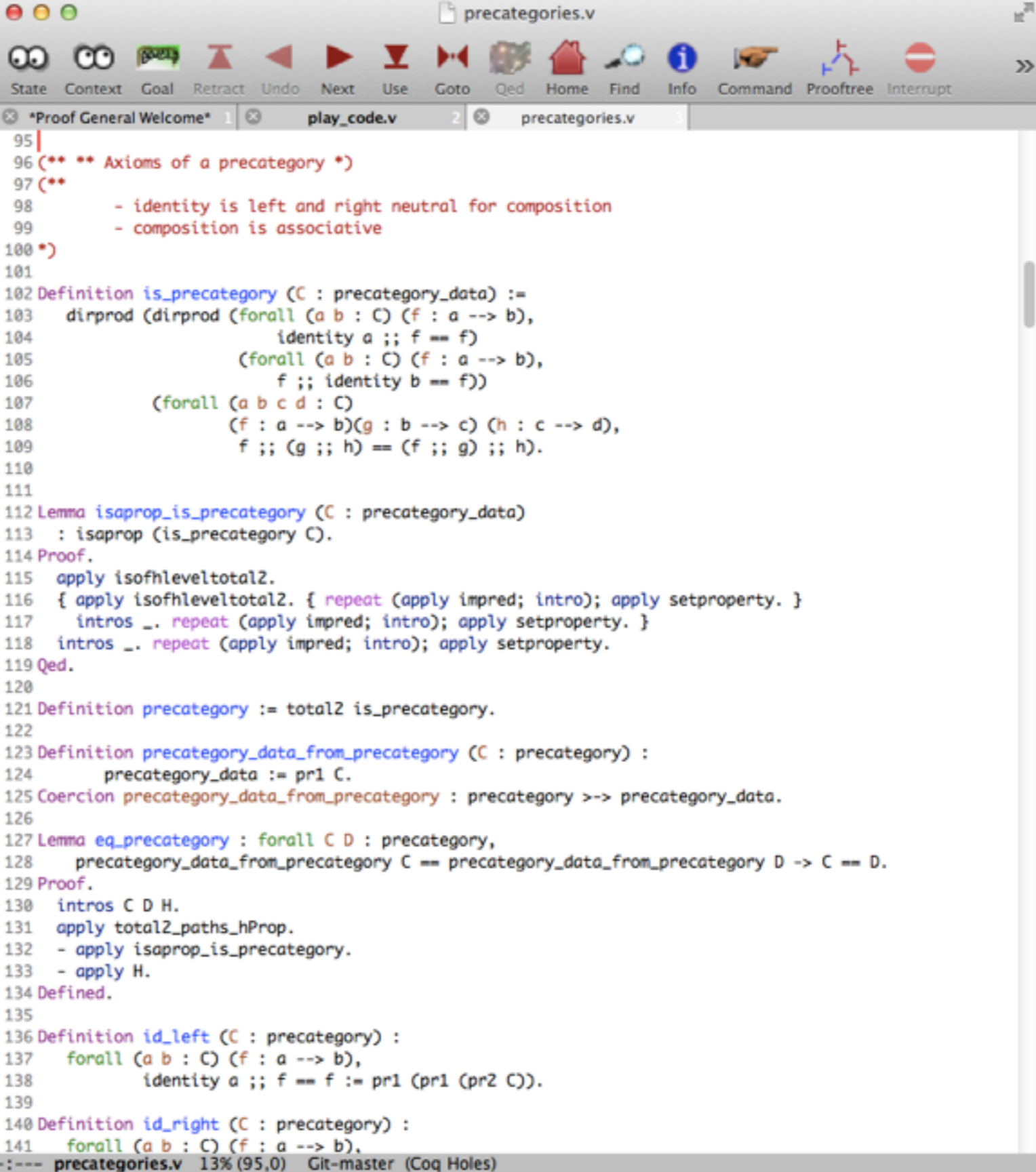
While it is possible to treat it from the position of computer science as well - as a description and possibly an implementation of an algorithm that would transform sentences of the UniMath language that pass the verification by the proof checker into inferences of the Zermelo-Fraenkel theory, at the moment we aim at a different realization of the fourth condition.



We expect the fourth condition to be realized originally in the form of a series of papers published in mathematical journals. These papers will provide not only a construction of a univalent model of the UniMath language but also tools that can be used to extend and modify this model to allow both for a step by step addition of new features and for a global restructuring of the language such as will be needed to upgrade from CIC to a cubical type theory.

Working on this papers is my main occupation at present.

Here is a screenshot of a session in proof assistant Coq with the code from one of the UniMath files.



```
precategories.v
State Context Goal Retract Undo Next Use Goto Qed Home Find Info Command Prooftree Interrupt
*Proof General Welcome* 1 play_code.v 2 precategories.v
95 |
96 (** ** Axioms of a precategory *)
97 (**
98     - identity is left and right neutral for composition
99     - composition is associative
100 *)
101
102 Definition is_precategory (C : precategory_data) :=
103   dirprod (dirprod (forall (a b : C) (f : a --> b),
104                   identity a ;; f == f)
105           (forall (a b : C) (f : a --> b),
106             f ;; identity b == f))
107   (forall (a b c d : C)
108     (f : a --> b)(g : b --> c)(h : c --> d),
109     f ;; (g ;; h) == (f ;; g) ;; h).
110
111
112 Lemma isaprop_is_precategory (C : precategory_data)
113   : isaprop (is_precategory C).
114 Proof.
115   apply isofhleveltotal2.
116   { apply isofhleveltotal2. { repeat (apply impred; intro); apply setproperty. }
117     intros _ . repeat (apply impred; intro); apply setproperty. }
118   intros _ . repeat (apply impred; intro); apply setproperty.
119 Qed.
120
121 Definition precategory := total2 is_precategory.
122
123 Definition precategory_data_from_precategory (C : precategory) :
124   precategory_data := pr1 C.
125 Coercion precategory_data_from_precategory : precategory -> precategory_data.
126
127 Lemma eq_precategory : forall C D : precategory,
128   precategory_data_from_precategory C == precategory_data_from_precategory D -> C == D.
129 Proof.
130   intros C D H.
131   apply total2_paths_hProp.
132   - apply isaprop_is_precategory.
133   - apply H.
134 Defined.
135
136 Definition id_left (C : precategory) :
137   forall (a b : C) (f : a --> b),
138     identity a ;; f == f := pr1 (pr1 (pr2 C)).
139
140 Definition id_right (C : precategory) :
141   forall (a b : C) (f : a --> b),
142   f ;; identity b == f := pr2 (pr2 C).
143
144 :--- precategories.v 13% (95,0) Git-master (Coq Holes)
```







