# A Coq toolkit for graph theory.

Jean Duprat

April 27, 2001

LIP, Ecole Normale Supérieure de Lyon.

**Abstract**

As an illustration of the Curry-Howard's isomorphism, it is possible to express both mathematical and computational aspects of the graph theory using the same formalism. In this paper, we give what can be the very beginning of such a presentation. The proof assistant Coq has been used as a guideline all along this work.

## 1 Introduction.

Graph theory has been largely developed during the last century. It appears as a branch of mathematics, with definitions, theorems and proofs [1]. But, since it is constantly used in computer science, implementations with data structure descriptions, algorithms and complexity measures, often face the mathematical formulae. But the consistency between the two languages is left to the reader cleverness. As an illustration of the Curry-Howard's isomorphism, it is possible to express both mathematical and computational aspects using the same formalism. We give the very beginning of such a presentation using the proof assistant Coq [3] as a guideline all along this work. We present successively basics, inductive definitions of graphs, paths and trees. In the section about graphs, we put the spot on dependent types as a good framework for defining graphs. In the following section, we show how refined structures as walk, trail and path are implemented by inductive definitions. We close by a discussion about the proof that a tree is an acyclic connected graph which illustrates the need of inversion lemmas.

## 2 Previous works.

In 1994, Chou [4] formalized directed graphs in higher order logic, but its approach was a non constructive one. The constructive way was followed later, for example by Yamamoto, Nishizaki, Hagiya and Toda [5], they proved theorems of planar graph theory using HOL. Their formalism is close to ours, but their

goal was only the proof of one theorem. In his PhD dissertation, Rouyer [?] used the first version of Coq to implement the Kruskal algorithm. Since graph theory is often used, we are sure that a lot of other implementations of a part of the theory can be found in many works about semantic, logic, vision ...

Our goal is to show that a constructive logic gives a formalism that is no less convenient than the mathematical usual one to describe graph theory. But, it reduces the gap between theory and computer implementation and, so it increases the consistency. This work has begun with C.Renard during his training period at the laboratory LIP of the ENS Lyon.

# 3 Axioms and basics.

## 3.1 Vertex.

The vertex is the primitive notion in graph theory. We assume that there exists an infinite number of vertices but we state an axiom of decidability of the equality between two vertices. Another choice is possible, saying that the vertices are indexed by natural numbers, so that the decidability of the equality between vertices is a consequence of the one between naturals.

### 3.1.1 Set of vertices.

A set is a non constructed object. Say that a vertex $v$ belongs to the set of vertices $V$ is a predicate; in the text, we will write it with the mathematical notation $v \in V$, but in Coq it is noted as the predicate $(V\, v)$. In absence of other information, the belonging relation is not decidable.

We set the extensionality axiom for proving equality between sets. We define the usual operations on sets : union, intersection, difference and inclusion. Special notations are used for empty set $\emptyset$ and singleton $\{v\}$.

### 3.1.2 List of vertices.

A list is a constructed object, inductively defined by the two constructors *nil* and *cons*. Because of the decidability of the equality of vertices, the occurrence of a vertex in a list of vertices is also decidable.

## 3.2 Arc.

An arc is defined as a pair of vertices. There is no structural difference between an arc and a list of two vertices, but, due to its frequent use, we prefer a specific definition with an immediate access to its two ends. In the following we will note the arc of tail $x$ and head $y$ : $\overrightarrow{xy}$ . The decidability of the equality between two arcs is clearly a consequence of the one upon vertices.

### 3.2.1 Set of arcs, list of arcs.

We use the same generic definition of set to define set of arcs as well as set of vertices. Similarly, we use the same structure of list parameterized by the type of Arc instead of the one of Vertex.

## 3.3 Edge.

An edge is defined as a set of two reciprocal arcs : $\{ \overrightarrow{xy} \quad \overrightarrow{yx} \}$. When there is no risk of ambiguity, we note an edge $xy$ . With such a definition, an edge is not a constructed object but, due to the finite number of possible cases, the equality between edges is decidable. Constructed definition of edge is possible, using either two vertices or two arcs, but the difficulty appears with the equality that has to assume $xy = yx$ . The nearest notion of an edge is an equivalence class of arcs by the symmetry relation, and the equality is the congruence relation. Recent works upon quotient by an equivalence relation sets and constructive sets will permit to improve this delicate point.

# 4 Graphs.

## 4.1 Inductive definition.

A graph is built upon a set of vertices $V$ and a set of arcs $A$ using one of the following rule. It is :

- the empty graph upon the empty set of vertices and the empty set of arcs;
- from a graph upon $V$ and $A$, adding a new vertex $x$, $x \notin V$, we obtain a graph upon $V \cup \{x\}$ and $A$;
- from a graph upon $V$ and $A$, adding a new edge $xy$ so that $x \in V$, $y \in V$, $xy \cap A = \emptyset$, we obtain a graph upon $V$ and $A \cup xy$;
- from a graph upon $V$ and $A$, with $V'$ a set of vertices extensionally equal to $V$ and $A'$ a set of arcs extensionally equal to $A$, we obtain a graph upon $V'$ and $A'$.

## 4.2 Constructors.

### 4.2.1 Empty graph.

The empty graph is the unique inhabitant of the graph upon the empty sets of vertices and edges.

### 4.2.2 Adding a vertex.

The added vertex is, by construction new and isolated in the obtained graph.

### 4.2.3 Adding an edge.

By construction, the added edge has its ends in the set of vertices. There is a redundancy in the definition between the condition $xy \cap A = \emptyset$ and the construction of the new set of arcs $A \cup xy$ in which we add both $\overrightarrow{xy}$ and $\overrightarrow{yx}$. A minimal but not symmetric condition such $\overrightarrow{xy} \notin A$ would be sufficient. The pro and cons arguments are the following, a redundant definition increases the number of goals to prove when you build the object, but it increases in the same time the number of direct properties deduced from the object when it resides in the hypothesis, so the balance is difficult to establish. We will find again the same debate when we will study the paths.

### 4.2.4 Compatibility with extensional equality of sets.

This constructor can look like a trick. On one hand, it is very useful since it offers an independence between the construction of the graph and the definition of the sets using classical operations on sets. On the other hand, it forbids the usual tactics of inversion since they loop upon such a constructor.

To the first argument, we say that it is only a point of view. If you judge that it is not a constructor, you can put this compatibility rule as an axiom. The result is quite similar.

To the second, we say that we cannot evitate to have to prove inversion lemmas by hand. Due to the fact that there are multiple means to built a graph, to backtrack of one step is not, in general, a deterministic operation. As we see further, we can prove under conditions, the obtention of a graph with one of these first three constructors, but the result is only obtained modulo this compatibility with the extensional equality on sets.

## 4.3 Comments.

In such an inductive definition, the mathematical definition of graph as a pair $(V, A)$ corresponds to the dependent type of graph upon $V$ and $A$. An inhabitant of this type is a construction, step by step, of the graph, adding vertices one by one, and edges, one by one too. The constraints are that every added object has to be new, and an edge can be added only when its two ends are already set. From a mathematical graph, we have in general, several constructed graphs which differ only by the order in which the vertices and the edges are chosen, but they inhabit the same type. Note also that some types have no inhabitants since their sets of vertices and arcs are inconsistent, either if an arc has one of its ends not in the set of vertices, or if an arc has not its symmetric version in the set of arcs.

## 4.4 Consequences.

From the inductive construction of a graph we can deduce :

- a list of the vertices (in the order of the construction) where each vertex has only one occurrence,
- the number of vertices, i.e. the order of the graph,
- the list of neighbors of a vertex and its degree,
- a list of arcs with unique occurrence,
- the number of edges, i.e. the size of the graph,
- the property that if an edge $xy$ occurs in $A$, its ends $x$ and $y$ belong to $V$;
- the decidability if or not a vertex belongs to $V$ or an arc belongs to $A$.

## 4.5 Inversion lemmas.

### 4.5.1 Empty set of vertices.

If the set of vertices $V$ is empty, then a graph on $V$ and $A$ is the empty graph and consequently the set of arcs is also empty.

### 4.5.2 Subtracting a vertex.

In a graph on $V$ and $A$, if $x$ is an isolated vertex $x \in V$, $\forall y \in V\ xy \cap A = \emptyset$, then there exists a graph upon $V \setminus \{x\}$ and $A$.

### 4.5.3 Subtracting an edge.

In a graph on $V$ and $A$, if $xy$ is an edge of $A$, there exists a graph on $V$ and $A \setminus xy$.

### 4.5.4 Proofs.

The proofs of these lemmas are done by elimination of the structure of the graph. These proofs use equality on sets, but, thanks to the fourth constructor, a rewriting of the sets can be applied into the goals.

## 4.6 Variants.

### 4.6.1 Unlooped graph.

It is easy to define in the same manner a graph with no loops. It suffices to modify the constructor "adding an edge" with the extra condition $x \neq y$ .

#### 4.6.2 Directed graph.

To obtain an inductive definition of a directed graph, we supersede the constructor "adding an edge" by the constructor "adding an arc" defined as followed:

- from a directed graph on $V$ and $A$, by adding the arc $\overrightarrow{xy}$ if the constraints $x \in V$, $y \in V$, $\overrightarrow{xy} \notin A$ are verified, we obtain a directed graph on $V$ and $A \cup \{\overrightarrow{xy}\}$;
- the other constructors are unchanged.

#### 4.6.3 ... is a ...

Some constructions can be trivially deduced from other ones. They lead to lemmas, for example "unlooped graph is a graph" or "graph is a directed graph" where the proof builds the goal by structural decomposition of the hypothesis where each case is trivially proved. This correspondence is usually expressed in mathematics with an inclusion relation which is more imprecise.

#### 4.6.4 From ... to ...

Other constructions are not so trivial, and require some choices. For example, we can build "from a graph to an unlooped graph" by removing all the loops, or "from a directed graph to a graph" by symmeterizing all the arcs on their first occurrence and skipping the second occurrence when it exists.

# 5 Paths.

## 5.1 Directed paths.

#### 5.1.1 Walk.

Given a set of vertices $V$ and a set of arcs $A$, a directed walk from an head vertex to a tail vertex following a list of vertices and a list of arcs is defined inductively by :

- the null walk relates each vertex $x$, $x \in V$, to itself following null lists ($nil$) of vertices and arcs;
- if a walk relates $y$ to $z$ with the list $vl$ of vertices and $al$ of arcs, if $x \in V$ such that $\overrightarrow{xy} \in A$, then there is a walk relating $x$ to $z$ with the list ($cons\ y\ vl$) of vertices and ($cons\ \overrightarrow{xy}\ al$) of arcs.

#### 5.1.2 Trail.

Given a set of vertices $V$ and a set of arcs $A$, a directed trail from an head vertex to a tail vertex following a list of vertices and a list of arcs is defined inductively by :

- the null trail relates each vertex $x$, $x \in V$, to itself following null lists ($nil$) of vertices and arcs;
- if a trail relates $y$ to $z$ with the list $vl$ of vertices and $al$ of arcs, if $x \in V$ such that $\overrightarrow{xy} \in A$ and $\overrightarrow{xy} \notin al$, then there is a trail relating $x$ to $z$ with the list $(cons\, y\, vl)$ of vertices and $(cons\, \overrightarrow{xy}\, al)$ of arcs.

### 5.1.3 Path.

Given a set of vertices $V$ and a set of arcs $A$, a directed path from an head vertex to a tail vertex following a list of vertices and a list of arcs is defined inductively by :

- the null path relates each vertex $x$, $x \in V$, to itself following null lists ($nil$) of vertices and arcs;
- if a path relates $y$ to $z$ with the list $vl$ of vertices and $al$ of arcs, if $x \in V$ such that $y \notin vl$, $(x \in vl \Rightarrow x = z)$ , $\overrightarrow{xy} \in A$ and $\overrightarrow{xy} \notin al$, then there is a path relating $x$ to $z$ with the list $(cons\, y\, vl)$ of vertices and $(cons\, \overrightarrow{xy}\, al)$ of arcs.

### 5.1.4 Cycles.

The following properties are defined :

- a closed walk is a walk from $x$ to $x$,
- a closed trail is a trail from $x$ to $x$,
- a cycle is a path from $x$ to $x$.

You find in [4] an inductive definition of cycles. But it needs to replace an arc by a couple of to contiguous arcs. The operation of removing an arc from a list is not simple. We have prefered to see a cycle like a property of a path.

### 5.1.5 Redundancy.

These definitions of walk, trail and path contain several redundant informations, indeed the tail is the last element of the list of vertices or the list of vertices can be deduced from the list of arcs. However, according to your interest, if it is more about the ends, or more about the list of vertices or more about the list of arcs, you can access directly to these informations without having to extract them from other informations. So, we think that the benefit is greater than the drawback.

### 5.1.6 Comments.

Usually, the notion of directed path is used in a directed graph, but since these definitions do not depend on the construction of the graph, we define them only from the set of vertices $V$ and the set of arcs $A$.

## 5.2 Undirected paths.

### 5.2.1 With constructed edges.

Using constructed edges and lists of them, it is easy to rewrite the previous definitions and obtain inductive constructions of undirected walks, trails, paths and cycles. The difficulty stands in the test if an edge belongs to a list or not since the equality is not the usual one but the equality modulo the symmetry as it was discussed in the chapter 1.3.

### 5.2.2 With arcs.

As we have defined undirected graphs using arcs, we can define undirected paths using list of arcs. The modifications to the previous definitions are in the conditions :

- we replace $\overrightarrow{xy} \in A$ by $\overrightarrow{xy} \in A$ and $\overrightarrow{yx} \in A$;
- we replace $\overrightarrow{xy} \notin al$ by $\overrightarrow{xy} \notin al$ and $\overrightarrow{yx} \notin al$;
- note that it is not useful to store in $al$ the symmetric of $\overrightarrow{xy}$, since we have verified its properties and due to the fact that the notion of path being naturally directed from head to tail, there is no ambiguity in the reading of the list $al$.

## 5.3 Properties.

### 5.3.1 Is a.

The proofs that a trail is a walk, a path is a trail and a path is a walk are trivial.

### 5.3.2 Characterizations.

The lemmas that a trail is a walk without repetitions of arcs and a path is a trail without repetitions of vertices are easy to prove.

### 5.3.3 Extraction.

The proof of the lemma "from walk to path", saying that from a walk, we can extract a path, implements the algorithm of extraction. It requires more work to be established.

### 5.3.4 Concatenation.

If there are a walk from $x$ to $y$ and a walk from $y$ to $z$, there is a walk from $x$ to $z$ and the proof builds it by appending the lists of vertices and the lists of arcs.

#### 5.3.5 Reverse.

From a path from $x$ to $y$, it is possible to build a path from $y$ to $x$ using the reversed lists. However, note that the list of vertices needs some care and is not immediately the reversed list since the tail belongs to the list and not the head.

# 6 Trees.

## 6.1 Acyclic graphs.

### 6.1.1 Inductive definition.

An acyclic graph is built upon a set of vertices $V$ and a set of arcs $A$ by :

- the empty acyclic graph upon the empty set of vertices and the empty set of arcs;
- from an acyclic graph upon $V$ and $A$, adding a new vertex $x$, $x \notin V$, we obtain an acyclic graph upon $V \cup \{x\}$ and $A$;
- from an acyclic graph upon $V$ and $A$, adding a new pendant vertex $x$ and its edge $xy$ so that $x \notin V$, $y \in V$, $xy \cap A = \emptyset$, we obtain an acyclic graph upon $V \cup \{x\}$ and $A \cup xy$;
- from an acyclic graph upon $V$ and $A$, with $V'$ a set of vertices extensionally equal to $V$ and $A'$ a set of arcs extensionally equal to $A$, we obtain an acyclic graph upon V$'$ and $A'$.

### 6.1.2 Properties.

- Clearly, we prove the lemma "an acyclic graph is a graph" by construction.
- The lemma : a cycle in an acyclic graph is reduced to a single vertex, is proved by structural elimination and considerations upon degrees of the vertices.

## 6.2 Connected graphs.

### 6.2.1 Inductive definition.

A connected graph is built upon a set of vertices $V$ and a set of arcs $A$ by :

- the connected graph upon the singleton $\{x\}$ as the set of vertices and the empty set of arcs;
- from a connected graph upon $V$ and $A$, adding a new pendant vertex $x$ and its edge $xy$ so that $x \notin V$, $y \in V$, $xy \cap A = \emptyset$, we obtain a connected graph upon $V \cup \{x\}$ and $A \cup xy$;

- from a connected graph upon $V$ and $A$, adding a new edge $xy$ so that $x \in V$, $y \in V$, $xy \cap A = \emptyset$, we obtain a connected graph upon $V$ and $A \cup xy$;
- from a connected graph upon $V$ and $A$, with $V'$ a set of vertices extensionally equal to $V$ and $A'$ a set of arcs extensionally equal to $A$, we obtain a connected graph upon $V'$ and $A'$.

#### 6.2.2 Properties.

- Clearly, we prove the lemma "a connected graph is a graph" by construction.
- In a connected graph, there always exists a path between two vertices, proved by structural elimination.
- As we shall see in the following, some inversion lemmas are proved.

### 6.3 Trees.

#### 6.3.1 Inductive definition.

A tree is built upon a set of vertices $V$ and a set of arcs $A$ by :

- the tree reduced to its root upon the singleton $\{r\}$ as the set of vertices and the empty set of arcs;
- from a tree upon $V$ and $A$, adding a new pendant vertex $x$ and its edge $xy$ so that $x \notin V$, $y \in V$, $xy \cap A = \emptyset$, we obtain a tree upon $V \cup \{x\}$ and $A \cup xy$;
- from a tree upon $V$ and $A$, with $V'$ a set of vertices extensionally equal to $V$ and $A'$ a set of arcs extensionally equal to $A$, we obtain a tree upon $V'$ and $A'$.

#### 6.3.2 Properties.

- We prove the lemma "a tree is a graph" by construction.
- We similarly proved "a tree is an acyclic graph" and "a tree is a connected graph".

#### 6.3.3 Reciprocal theorem.

- The reciprocal theorem, "an acyclic and connected graph is a tree" needs more attention.

- Informally, we can say that an acyclic graph is built by the two constructors : adding a vertex, adding a pendant vertex, and a connected graph by the two constructors : adding a pendant vertex, adding an edge, even when a tree is built with the unique constructor adding a pendant vertex. Considering that adding an isolated vertex into an acyclic graph creates a new connected component that never will be bound to the other components, and that adding an edge into a connected graph creates a cycle which will never disappear, we can build a graph which is both acyclic and connected only using the shared constructor adding a pendant vertex, therefore we build a tree. But, this is not a proof.

- To prove the theorem, we have to break the symmetry of the hypotheses. We can follow one of these two ways : either by structural elimination upon the acyclic hypothesis and inversion upon the connected hypothesis, or by elimination upon connected and inversion upon acyclic. Each one of these ways amounts to follow the construction of one hypothesis, and use the second hypothesis to eliminate whatever does not lead to a tree.

- We have followed the first way and used the inversion lemmas : if there is an isolated vertex in a connected graph, the graph is reduced to the single vertex and if there is a pendant vertex $x$ bound to $y$ in a connected graph upon $V$ and $A$, there exists a connected graph upon $V \setminus \{x\}$ and $A \setminus xy$.

#### 6.3.4 Discussion.

Since the last proof is not as easy as we hoped, we wonder if another choice for defining acyclic, connected and trees does not lead to simpler proofs. The usual presentation by characteristic properties is to say that :

- an acyclic graph is a graph where every cycle is reduced to a single vertex,

- a connected graph is a graph in which two vertices are bound by a path,

- a tree is a both acyclic and connected graph.

So, there is nothing to prove since the lemma is true by definition. But the problems appears when we want to prove any property upon trees (or upon acyclic graphs or upon connected graphs), like for example a n-vertex tree has n-1 edges. We cannot proceed by structural elimination, the unique constructed structure on which to base the elimination would be the graph, but every graph constructors does not keep the tree property. We have to find invariant properties true upon graphs and prove them. In the example, we can prove that a n-vertex graph with more than n-1 edges has a cycle and that in a n-vertex graph with less than n-1 edges there are two vertices with no path relating them.

#### 6.3.5 Comparison.

As a rule we can say that :

- defining an object by its characteristic property leads to easy proof when the object appears like the goal of the lemma, For example, given a set of vertices $v$ and a set of arcs $a$, it is often simpler to prove the existence of a path between a couple of vertices of $v$ than to built a connected graph upon $v$ and $a$.
- at the opposite, defining an object by its construction leads to easier proofs when the object is an hypothesis of the lemma, since we can do an elimination upon the object.

So, we conclude in favor of the construction definitions.

# 7 Conclusion.

This work is an attempt to unify mathematical and computer programming aspects of the graph theory under the same formalism. The goal is to propose a library of the basic concepts offering a simple and convenient framework to write proofs as well as programs. But, only the very beginning of this work is now done, it is stored as a Coq contribution into a first version. We hope to develop this version in the future.

It is however interesting to see that such a change of point of view upon a theory, without bringing any novel concept, involves an exchange between definitions and characteristic properties. But, choosing what will be simple is not easy, and simplicity or convenience are not qualities that can be measured. With the proof assistant Coq, each bad choice induces lines and lines of tactics in the proof development. So, Coq works like a communication cord and warns when the development is possibly needlessly complex, so it is time to question the choices and eventually to backtrack into the work.

# References


[1] R.Balkrishnan, K.Ranganathan, A textbook of graph theory, Springer.

[2] B.Barras, S.Boutin, C.Cornes, J.Courant, Y.Coscoy, D.Delahaye, D.de Rauglaudre, JC.Filliatre, E.Gimenez, H.Herbelin, G.Huet, H.Laulhere, C.Munoz, C.Murthy, C.Parent-Vigouroux, P.Loiseleur, C.Paulin-Morhing, A.Saïbi, B.Werner, The Coq Proof Assistant, Reference Manuel, INRIA.

[3] Ching-Tsun Chou. A formal theory of undirected graphs in higher-order logic. Vol 859, Lecture Notes in Computer Science, Vlume 859, pp 144-157, Springer-Verlag, 1994.

[4] M.Yamamoto, S.Nishizaki, M.Hagiya and Y.Toda, Formalization of Planar Graphs, High Order Logic Theorem Proving and its Applications, Lecture Notes in Computer Science, Springer-Verlag, Volume 971, pp 369-384, 1995.

[5] J.Rouyer, Developpements d'algorithmes dans le Calcul des Constructions. PhD thesis. Institut National Polytechnique de Lorraine, 18 mars 1994.