Finite groups representation theory with Coq

Sidi Ould Biha

Finite groups representation theory with Coq

Sidi Ould Biha

INRIA Sophia Antipolis,
INRIA Microsoft Research Joint Centre
Sidi.Biha@sophia.inria.fr

Abstract. Representation theory is a branch of algebra that allows the study of groups through linear applications, i.e. matrices. Thus problems in abstract groups can be reduced to problems on matrices. Representation theory is the basis of character theory. In this paper we present a formalization of finite groups representation theory in the Coq system that includes a formalization of Maschke's theorem on reducible finite group algebra.

Key words: Representation theory, Maschke's theorem, linear algebra, Coq, SSReflect

1 Introduction

The use of proof assistants for the formalization of mathematical theories has increased considerably in recent years. Success stories like the formal proofs of the Four Colour theorem [1] or the prime number theorem [2] have shown that formal proof systems have reached the age of maturity. After these successes, ambitious projects were launched, for example the Flyspeck [3] project which aims to develop a formal proof of Kepler's conjecture. Projects such as the C-CoRN [4] have developed large repositories of mathematical formal proof libraries, but the number of formal mathematics libraries remains low compared to the number of libraries developed in Computer Algebra System (CAS) like Mathematica [5] or GAP [6]. This is one of the reasons for the limited number of users of formal proof systems, especially among mathematicians.

This work is a part of the Mathematical Components project [7] which aims to develop a formal proof of the Feit-Thompson theorem [8]. Finite group representation theory is among the large variety of mathematical theories covered by the proof of the Feit-Thompson Theorem.
This paper presents a formalisation of finite group representation theory and generic libraries for linear algebra : theory of finitely generated modules over algebras and fields. A formal proof of the Maschke theorem was also developed. This is done using the SSReflect [9] extension of the Coq proof assistant [10, 11].

[1] This research work was funded by the Microsoft Research INRIA joint centre

The paper is organized as follows. In Section 2 we give an introduction to finite group representation theory and show how it is linked to module theory. In Section 3, we present the Mathematical Components project, the Coq extension SSReflect and the project libraries we reused in our development. Finally, in Section 4, we present the two components of this development : the linear algebra and representation libraries.

2 Representations theory

Finite group representation theory studies the structure of a finite group by presenting it as a matrix. For example, the symmetric group of index three S_3 can be represented as the group of the isometries of an equilateral triangle. The symmetric group of index four S_4 can be represented as the group of rotations of a cube. This same technic can be used to study other algebraic structures likes associative algebra and Lie algebra. Historically the theory was introduced in the second half of the nineteenth century by Frobenius to solve problems from Galois theory. It was largely developed afterwards to be a basic tool for the classification of finite groups and an important part of the proof of the Feit-Thompson theorem. Representation theory is used in algebraic number theory through the class field theory. It is also used in the Langlands program [12], an active field of contemporary mathematical research.

Algebra representation : Given a field F, an integer n and an F-algebra A, a representation of A is an algebra homomorphism $\phi : A \to M_n(F)$, where $M_n(F)$ is the algebra of square matrix of size n and coefficients in F. Generally, A can be an associative algebra or Lie algebra.
In representation theory literature [13–15], a common way to study representations is to see them as modules. In Isaacs book [13], which is the reference for representation theory for the proof of Feit-Thompson theorem, a theory of finitely generated modules over algebra is developped to introduce representation theory. A module over a finite algebra has also a structure of finite F-vector space, since for any F-algebra A, $F.1 = \{c1 | c \in F\}$ is a subalgebra of A. Thus a module over an F-algebra A is a F-vector space V with a right action of A on V such that for all $x, y \in A, v, w \in V$ and $c \in F$ the following properties hold :

- $(v + w)x = vx + wx$,
- $v(x + y) = vx + vy$,
- $(vx)y = v(xy)$,
- $(cv)x = c(vx) = v(cx)$,
- $v1 = v$

With this definition we have that for any F-algebra A, every representation of A has an A-module structure and conversely every A-module provides a representation of A. Thus we have an equivalence between representations and A-modules. The advantage of this approach is that many definitions and results on representations can be borrowed from module theory. With this equivalence we can

introduce some definitions on representations. In the following, A is an F-algebra and V is a representation (in others words an A-module) :

- A *subrepresentation* of V is an A submodule W of V or an F subspace W of V which is stable under the action of A. A *subrepresentation* is also a representation.
- V is *irreducible* if its only submodules are 0 and V. It is *semisimple* if for every submodule $W \subseteq V$, there exists another submodule $U \subseteq V$ such that $V = W \oplus U$ in other words V is the direct sum of W and U.
- A representation or more generally a module is *semisimple* if it is the finite direct sum of *irreducible* submodules. These two last definitions are equivalent.

Finite group representation : Let G be a finite group, F a field and $GL(n, F)$ the multiplicative group of non-singular $n \times n$ matrices on F. An F-representation of G is a group homomorphism ρ from G to $GL(n, F)$. The integer n is called the degree of ρ. Thus a representation is a function $\rho : G \to GL(n, F)$ such that :

$$\rho(1_G) = I_n \qquad and \qquad \forall g\ h \in G\ \rho(gh) = (\rho g)(\rho h)$$

Group algebra is a key structure in representation theory. It links the definition above of group representation to that of algebra representation and modules theory. Given a finite group G and a field F, the group algebra $F[G]$ is the set $\{\sum_{g \in G} a_g g \ \mid \ a_g \in F\}$. This set has a structure of F-vector space and F-algebra. Indeed, the function that associates to any element g of G the element $\sum_{h \in G} a_h h$ with $a_g = 1$ and $a_h = 0\ if\ h \neq g$ embeds G into $F[G]$, so we can see G as a basis for $F[G]$. The addition and external multiplication are defined as follows :

$$\sum_{g \in G} a_g g + \sum_{g \in G} b_g g = \sum_{g \in G} (a_g + b_g) g \qquad \alpha \sum_{g \in G} a_g g = \sum_{g \in G} (\alpha * a_g) g$$

The $F[G]$ internal multiplication law is defined by considering the group multiplication :

$$(\sum_{g \in G} a_g g)(\sum_{h \in G} b_h h) = \sum_{k \in G} (\sum_{gh = k} a_g b_h) k$$

With this law and 1_G, $F[G]$ has a structure of an F-algebra. It follows that for any finite group G, an F-representation of G can be seen as the restriction on $G \subset F[G]$ of a representation of $F[G]$. Thus any group representation has an $F[G]$-module structure and vice versa.

An important result on finite group representation is the Maschke's theorem. It states that :

For any finite group G and a field F whose characteristic does not divide $\mid G \mid$, every representation ($F[G]$-module) is semisimple.

Maschke's theorem reduces the study of group representations into the study of *irreducible* representations. This is given by the fact that every group representation is the direct sum of *irreducible* representations. In order to know all the representations of a finite group, it suffices to know all its irreducible representations.

3 Mathematical Component project

In the classification of finite groups, the Feit-Thompson theorem is a central result. His proof revolutionized group theory not only by the techniques it introduces but also by its length. The original paper [8] is more than 250 page long and remains roughly the same despite all the efforts to simplify it. The verification of the paper proof took about a year for a team of specialists in group theory. More generally, several results in the theory of classification have been published in papers whose length reaches hundreds of pages. The formalization of these proofs is a real challenge for proof assistants. Based on the experience gained in the proof of the Four Colour theorem, the Mathematical Components project aims to develop a formal proof of Feit-Thompson theorem.

SSReflect

In the Mathematical Components project, the development enviroment is SSReflect, a Coq extension who was developed by G. Gonthier for the formal proof of the Four Colour theorem. SSReflect (for *Small Scale Reflection*) introduces a new language for tactics that eases the development of proof scripts. It allows the user to write more concise proof scripts than those written using the standard Coq tactic language. Another main feature is the generic reflection mechanism. In the Coq proof system, the default logic is intuitionistic. In this logic, logical propositions and boolean values are distinct. Logical propositions are objects of type `Prop` which is the carrier of intuitionistic reasoning. The boolean type is an inductive type with two values : true and false. These two structures are complementary. The first one makes it possible to have structured proofs by using natural deduction whereas the second makes it possible to perform computation. SSReflect introduces a generic reflection mechanism that allows to combine the best of the two views and to switch from the propositional version of a decidable predicate to its boolean version. More details on the SSReflect tactics language and the view mechanism are presented in the SSReflect manual [9].

Libraries

In the project, we have a large variety of libraries that gives definitions and properties for a variety of mathematical structures. In all this development, libraries are independent from the excluded middle and the choice axiom. The logical requirements are internalized in the structure definitions.

SSReflect includes, among others, the following libraries :

- `eqtype`: type with a a decidable equality which is equivalent to the Leibniz one.
- `choice`: type with choice operator.
- `fintype`: type with finite elements.
- `finfun`: type of function of finite domain.
- `bigops`: generic indexed "big" operations, like $\sum_{i=0}^{n} f(i)$ or $\max_{i \in I} f(i)$.
- `groups`: finite groups theory.
- `ssralg`: algebraic structures from abelian group to algebraic closed field.
- `matrix`: determinant theory and matrix decomposition (LUP decomposition).

The libraries include also results on finite groups theory like the Sylow theorems and the Cauchy-Frobenius lemma. The Cayley-Hamilton theorem about matrix and polynomial was also formalised. For more precise details on theses libraries we refer to [16, 17].

4 Formalization

The Feit-Thompson theorem covers a variety of different mathematical theories such as linear algebra and finite group theory. The work of formalization of such large theory requires an approach similar to software engineering. In this design process, the choice of which data structure to use to represent a mathematical concept is important. This choice must take into account the needs of genericity and reusability. The proof assistant Coq provides mechanisms such as dependent types and records, coercions or canonical structures that meet those needs.

Coq's dependent records [18] are useful to encode data types such as algebraic structures where we have a set of axioms and operations associated to the type of elements. They have been used in several algebraic hierarchy formalization such as [19] and [20]. Coercions provides a sub-typing mechanisms. They facilitate the development of generic theories and sharing of notations for abstract structures. This is very useful especially for the development of theory on algebraic structures where it is common to have inheritance : vector spaces are a sub-type of commutative groups and algebras are a sub-type of vector spaces and rings. Canonical structures allow the inference of a specific structure for a specific type. It works in the opposite direction to that of coercions. For example the matrix type can be equiped in a canonical way by a ring structure. With canonical structures, this structure can be implicitly infered by the system. When A and B are matrices, when writing the expression $A + B$, the system automatically infers that the $+$ corresponds to the additive group law of matrices. When m and n are integers, when writing the expression $m + n$, it automatically infers that the $+$ corresponds to the additive group law of integers. It is common in mathematical literature to let the reader infer from the context the corresponding operation.

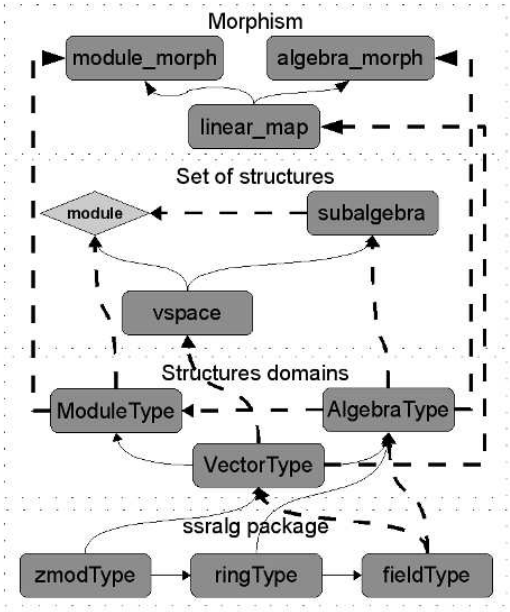

Fig. 1. The hierarchy for linear algebra

The use of `ssralg` requires that the domain type on which the algebraic structure is defined has a decidable equality and a choice operator. This two requirements are internalized in the structures `eqType` and `choiceType`. This means that in order to use the theories implemented by this library for a certain type, this axioms should be valid on the elements of the type.
In an intuitionistic type theory base proof assitant like Coq, this design approch gives a general quotient construction, like ideal quotient or canonical basis for finitely generated modules. It also allows the use of Coq's powerfull rewriting system thanks to the inclusion of the decidable equality in the Leibniz's one (Coq's default equality). The alternative to this approach is the use of `Setoids` [21] like what is done in [19] and [20]. The corresponding equivalence relation has then to be handled explicitly. This approach is costly especially when, as in our case, we deal with advanced mathematicals statements that involve several mathematical structures. Also, in the proof of the Feit-Thompson theorem, the algebraic structures handled are mainly finite groups or finite dimension vector spaces over finite fields or the algebraic number field. For these structures the decidable equality and the choice operator can be defined constructively.

4.1 Linear algebra

As already said in Section 2, finite group representation theory inherits many definitions and results from linear algebra. Our main motivation is the formaliza-

tion of finite group representation theory but the part on linear algebra can be used independently in other formalizations. In finite group representation theory, modules that are taken into consideration are all of finite dimension and are either defined over a field (finite or not) or on an algebra. We formalized a linear algebraic structures hierarchy that covers the theories of :

- finite dimension vector space
- finite dimension algebra
- finitely generated module over an algebra

The development is built on top of the algebraic hierarchy given by the `ssralg` library. It consists of three layers. The first layer defines the interfaces and provides the generic theory for the domain of the considered structures. The second layer defines the interfaces and provides the generic theory for the special sets associated with the domain type structures of the first layer: sub-vector spaces, sub-algebras and sub-modules. The third layer is dedicated to morphisms of structures: linear applications, algebra and module morphisms.
Figure 1 provides an overview of the composition of these layers. In this figure, the plain arrows represent the sub-typing, for example `AlgebraType` is a subtype of `VectorType` and `ringType`. The dashed arrows represent type dependency, for example `VectorType` type depends on a `fieldType`.

Structures domains : An abstract algebraic structure is a combination of a representation type (domain), constants and operations on this type, and axioms satisfied by this constant and operations. For example, a group is a set G together with a constant e, an internal operation $*$ and a list of axioms (neutral element, associativity of $*$ ...). Also, algebraic structures are defined according to a hierarchical scheme. A vector space is a commutative group that has an external law which acts from a field. An algebra is the combination of a vector space and a ring structure with additional axioms. For the definition of our domains of algebraic structures, we apply the generic method introduced in the `ssralg` library. This method gives a design pattern for the definition of such domains. It is mainly motivated by problems of packaging (inheritance and sharing) and performances. For example, Figure 2 gives the interface of the finite dimension vector space.

We use the Coq `Module` system to create a separate name space for the F-vector space structure and avoid any clash of definitions, since the declaration of other algebraic structures follows the same scheme. In Figure 2, `zmodType` represents the type of commutative group that also has a decidable equality on its elements and a choice operator. The module `Equality` packages types with a decidable equality and the module `Choice` packages types with a choice operator. In this figure, different structures are defined :

- The `mixin_of` structure packages the additional operator (external law), the constants (basis) and the axioms needed by a commutative group to be an F-vector space.

```
Module Vector.
Section VectorTypeDef.
Variable F : fieldType.

Structure mixin_of (V : zmodType) : Type := Mixin {
  mul : F -> V -> V;
  _ : forall a b u, mul a (mul b u) = mul (a * b) u;
  _ : forall u, mul 1 u = u;
  _ : forall a, {morph mul a : u v / u + v};
  _ : forall a b u, mul (a + b) u = (mul a u) + (mul b u);
  basis : seq V;
  _ : forall s, \sum_(i < size basis) mul s`_i basis`_i = 0 -> forall i,
      i < size basis -> s`_i = 0;
  _ : forall v, exists s, v = \sum_(i < size basis) mul s`_i basis`_i
}.
Structure class_of (V : Type) : Type := Class {
  base :> Zmodule.class_of V;
  ext :> mixin_of (Zmodule.Pack base V)
}.
Structure type : Type :=
 Pack {sort :> Type; _ : class_of sort}.
...
Definition eqType cT := Equality.Pack (class cT) cT.
Definition choiceType cT := Choice.Pack (class cT) cT.
Coercion zmodType cT := Zmodule.Pack (class cT) cT.

End VectorTypeDef.
End Vector.
...
Notation "a *: v" := (mulv a v) (at level 40) : ring_scope.
```

Fig. 2. Vector space interface

- The `class_of` structure packages all the theories needed by a representation type (here `V : Type`) to be an F-vector space. The first projection `base` is a proposition that states that `V` has a structure of commutative group. The second projection `ext` is a proposition that states that the `zmodType` built on `V` (with the call of the function `Zmodule.Pack`) with the structure `base` has the additional structure to be an F-vector space.
- The `type` structure corresponds to the F-vector space interface. In this structure, the declaration `sort :> Type` makes the `sort` projection a *coercion* from `type` to `Type`. This form of explicit sub-typing allows any `V : vecType` to be used as a `Type`, e.g., the declaration `x : V` is understood as `x : sort V`. It is useful for getting generic theorems for abstracts structures.
- The declarations `eqType`, `choiceType` and `zmodType` at the end of the `Module` define the inheritance rules.

- The last line of the listing declares `*:` as a notation for the external law of the vector space.

The other structures of the hierarchy, algebra and finitely generated modules over algebra are defined following the same design pattern. The only difference comes from the composition of the structures `mixin_of` and `class_of`.

Set of Structures : In finite group representation theory, questions about relations between different representations of a given group are very frequent. Is it true that a given representation of a group is irreducible? Is it true that two representations are equivalent or complementary? As we have already said, these questions can be reduced to questions on relations between modules defined on an algebra. In the case of representations, the algebra will be the group algebra. We have thus formalized a theory of sub-vector spaces. The algebraic structures we are interested in are algebras and modules of algebras. They have a structure of vector space. Thus, the corresponding sub-structures are sub-vector spaces with an additional property on the internal multiplicative law (algebra) and the external law (module).

In a type theory framework, sub algebraic structure, likes sub-group or sub-space, are usually defined as a propositional or boolean predicate. For example, in the Coq system, they can be represented as a dependent structure with two elements : a propositional predicate and a closure property. The type of sub-groups of given group can be defined as follows :

```
Structure sub_group (G : group) : Type := SubGroup {
 set :> G -> Prop;
 is_sub_group : forall a b : G, set (a - b)
}.
```

The problem with this representation is that in order to have equality between sub-structures, we need an axiom of extensionality. In finite dimension vector space theory, there is no need for this axiom. A set of vectors of a finite dimension vector space always has a family of generators. If, in addition, this family is free then the set is a vector space. Conversely, every family of vectors defines a sub-space. Thus, a sub-space of finite dimension vector space can be represented by a list of vectors : the generators. Deciding the membership to a family of generators is equivalent to deciding if there is a solution for a linear system. To be able to view a set of vectors as a boolean predicate, we have added two axioms assuming that for all linear systems the problem of the existence of a solution is decidable :

```
Axiom system_dec : forall m n (F : fieldType),
  matrix F m n -> matrix F m 1 -> bool.
Axiom system_dec_ex : forall m n (F : fieldType) (A : matrix F m n) v,
 (exists vs, A *m vs = v) <-> (system_dec A v).
```

These two axioms can be removed once a procedure for solving linear systems is formalized. In the project libraries, we are not far from having one since the matrix LUP decomposition has already been formalized.

We define the type of sub-spaces of a vector space as a list of vectors with a predicate specifing that it is the canonical family of generators. We use `choose`, the choice operator, to quotient with the spanning set equality relation and then identify with a Leibniz equality all the families that generate the same vector space.

```
Variable (K : fieldType) (vT : vecType K).

Structure vspace : Type := VSpace {
 gf :> seq vT;
 _ : gf == choose [pred x | free_gf x && gf_eq gf x ] (basis_for_gf gf)
}.
```

In this definition `seq vT` is the type of lists over `vT`. The notation `==` corresponds to the decidable equality associated with the type `seq vT`. Thanks to canonical structures, Coq will automatically infer the corresponding equality. The function `choose` is provided by the `Choice` interface. It takes two parameters: a predicate and an element. It returns a "canonical" element that satisfies the given predicate if the element given as parameter satisfies already the parameter predicate, i.e. it is the witness that the predicate is satisfiable. In the definition above, we require that the list of vectors is equal to the result of the application of `choose` to the predicate that checks if a given family of vector is free (`free_gf`) and the set its generates is equal to the one generated by `gf` (relation `gf_eq`). The function `basis_for_gf` returns a free basis for a given family of generators. It proceeds by removing the dependent vectors.

After that, and in order to be able to view sub-vector spaces as extensional sets (functions of type `vT -> bool`) which are more practical for proofs, we have declared a coercion from the `vspace` type to `predPredType`. It is a generic interface for the type of boolean predicates provided by the SSReflect library `ssrbool` :

```
Coercion pred_of_vs :=
 (fun F (vT : vecType F) (V : vspace vT) => mem_gf V : _ -> _).
```

We constructively define sub-space operations likes the sum and intersection of two sub-spaces and the complement of a given sub-spaces. We also prove some membership properties for sub-vector spaces :

```
Lemma vs_mul : forall c v, c *: v \in V = ((c == 0) || (v \in V)).
Lemma eq_vsP : forall V1 V2,
  (forall v, v \in V1 = (v \in V2)) <-> (V1 == V2).
```

In this statement, the equality `=` stands for Coq standard equality between boolean values. The first lemma provides a rewriting rule for the membership of a product. The second lemma gives an equivalence between the extensional equality of sub-spaces and their decidable equality. It allows switching between the two views.

We define sub-algebras and sub-modules as boolean predicates over `vspace` :

```
Definition is_sub_algebra (V : {vspace aT}) :=
 forallb i : 'I_(\dim_V), forallb j : 'I_(\dim_V), (V`_i * V`_j) \in V.
Definition module (A : salgebra aT) (V : {vspace mT}) :=
```

```
forallb i : 'I_(\dim_A), forallb j : 'I_(\dim_V), (V`_j :* A`_i) \in V.
```

In these definitions, `forallb` is a notation for the boolean universal quantifier and `'I_(dim_V)` is the finite type of all integers less than `dim_V` the dimension of the sub-space `V`.

In order to check if a sub-space of an algebra is a sub-algebra we only need to check that the multiplication of any two elements of the basis is included in the basis. The definition of the module predicate is parametrized by a sub-algebra and not the whole algebra domain. Indeed, any module on an algebra has also the structure of module on every sub-algebra of the original algebra. The genericity will be useful when defining sub-group representations.

Morphisms : A linear application between two vector spaces V and W is a function $f : V \to W$ such that :

$$\forall a\ u\ v, f(a * u + v) = a * fu + fv$$

If V and W are of finite dimension, then every linear application from V to W can be represented as a matrix. Conversely, every $n \times m$ matrix defines a linear application. The functional view is more practical to handle when doing proofs. The matrix view gives a finite description, which is suitable for encoding, and inherits Leibniz equality and choice operator from the corresponding field of the vector spaces.

We represent linear applications as singleton type that contains a matrix. In order to be able to see them as functions, we define a coercion from the type of linear applications to that of functions :

```
Variable (K : fieldType) (vT wT : vecType K).

Inductive linear_map : Type :=
 LinearMap of (matrix K (size (basis wT)) (size (basis vT))).

Definition lmap_mx f := let: LinearMap M := f in M.

Definition fun_of_lmap := (fun K vT wT f v =>
 let fv := (@lmap_mx K vT wT f) *m (@mx_of_vec K vT [:: v]) in
 \sum_(i < size (basis wT)) fv i (Ordinal (ltnSn 0)) *: (basis wT)`_i).
```

The library also provides a constructor of linear applications (elements of type `linear_map`) from a function. It builds the matrix corresponding to the image of the basis of the domain according to the basis of the codomain.

```
Definition lmap_of_fun := (fun K (vT wT : vecType K) (f : vT -> wT) =>
 let m := size (basis wT) in let n := size (basis vT) in
 let M := \matrix_(i < m, j < n) (decomp (f (basis vT)`_j))`_i in
     LinearMap M).
```

In the library we have constructively defined the kernel of a linear application and the linear projection on a sub-space. The canonical vector space construction for the linear application type is also defined.

4.2 Representations

The main motivation of this work is the formalization of finite group representation theory. In this development, our main reference is the book by I. Martin Isaacs[13]. That is why we started our work by formalizing a theory of free modules on algebras and fields. We also found inspiration in the ideas presented in [15].

Definitions : The representation type is defined using the Coq dependent type record.

```
Variables (gT : finGroupType) (F : fieldType) (n : pos_nat).

Structure representation (A : {set gT}) : Type := RepPack {
 ro_ :> gT -> (matrix F n n);
 _ : {in A &, forall g h, ro_ (g * h) = ro_ g * ro_ h};
 _ : ro_ 1 = 1
}.
```

The definition is parametrized by a finite group domain type `gT`, a field `F` and a positive integer `n`. The representation type is defined for a given set `A` of the finite group domain `gT`. The first component of the structure is a function from the finite group domain `gT` to the type of square matrix of size `n` on `F`. The two other components state that a representation is a group morphism.
In this definition and thanks to the use of `Canonical Structures`, the structure of ring for matrix is automatically infered by Coq. This allow us to use the standard notations for ring structures : the ring multiplication `*` and neutral element `1` in the second part of the last two statements.

Group algebra : To link the above definition of representation to the module theory, the group algebra is defined. This is done using the SSReflect `finfun` library which contains a complete formalization of finite domain functions theory. For a finite group `G` and a field `F`, the group algebra $F[G]$ is defined as the type of functions of type `G -> F`.

```
Variables (F : fieldType) (G : finGroupType).
Notation Local "F ,[ G ]" := {ffun G -> F}.

Definition gA0 : F,[G] := [ffun g => 0].
Definition gA1 : F,[G] := [ffun g => if g == 1 then 1%R else 0].
Definition opprgA (v : F,[G]) : F,[G] := [ffun g => - (v g)].
Definition addrgA (v1 v2 : F,[G]) : F,[G] := [ffun g => (v1 g) + (v2 g)].
Definition mulvgA (a : F) (v : F,[G]) : F,[G] := [ffun g => (a * (v g))%R
    ].
Definition mulrgA (v1 v2 : F,[G]) : F,[G] :=
 [ffun g => \sum_(k : G) (v1 k) * (v2 ((k^-1) * g)%g) ].

Definition gAbasis : seq F,[G] :=
 map (fun g => [ffun k => if k == g then 1%R else 0]) (enum G).
```

The ring and vector space operations for this type are defined using the generic constructor `[ffun g => E]` which constructs the graph of the function that associates to `g` the expression `E`. We also define the associated sub-algebra structure for a sub-group of the original group domain. It is the sub-space of $F[G]$ generated by the elements of the sub-group.

Maschke's Theorem : Our library for representations theory includes a formal proof of Maschke's theorem. The Coq's statement of the theorem is the following :

```
Section Maschke.
Variables (gT : finGroupType) (G : {group gT}) (F : fieldType).
Variable (mT : modType F,[gT]).
Notation Local "|G|" := (#|G| %:R : F).
Notation Local "[F/G]" := (groupSAlg F G).
Hypothesis (HcardG : |G| != 0).

Theorem Maschke :
 forall V : {vspace mT}, module [F/G] V -> semisimple [F/G] V.
Proof.
...
Qed.
```

In this statement `[F/G]` is the notation for the `F[gT]` sub-algebra associated to the sub-group `G`. The proposition `semisimple` expresses the fact that every sub-module `W` of `V` has a direct complement.

The idea of the proof [13] is to take, for a sub module W of V, an F-linear projection on it. From this projection, we build a new $F[G]$-projection on it. The kernel of this new projection is an $F[G]$ sub-module of V and also a complement of W. In our formalization, the proof is 34 line long, the doube of the standard paper proof.

5 Related Works

In the proof assistant community, there has been some developments on linear algebra that cover part of what we have formalized. To our knowledge, none has tackled representation theory.

The set-theoretic Mizar Mathematical Library (MML) [22], which has the largest library of formal mathematics, contains formalizations of algebraic structures such as groups, rings, modules and real vector spaces. This structures are defined in various articles and by various authors.

In the Coq system, there are essentially two constructive algebraic hierarchy that have been developed. The first is the seminal Algebra repository [20], which constructs algebraic structures from monoids to modules. The second is the C-CoRN hierarchy [19], mainly devoted to a constructive formalisation of real numbers and including a proof of the fundamental theorem of algebra. Both are setoid based and have been proved difficult to extend with theories like linear or multilinear algebra.

6 Conclusion

The work we present here provides a formalization of finite group representation theory in the Coq system. It also include also a formal proof of the Maschke's theorem. It shows that the different components of our development work well together.To facilitate the reuse of this development, we used a modular approach. This is an important point especially for large formalizations such as the one we work on in the Mathematical Component project. The formalization of algebraic theory is not an easy task especially in terms of packaging and reuse. Coq's dependent types, `Coercions` and `Canonical Structure` have contributed to the achievement of this work. They provide a powerful mechanisms for formalizing abstract algebraic structures. SSReflect and its large libraries of formal proofs and theories have also been very useful. We have used several of these libraries and especially `ssrlag` for basic algebraic structures (groups, rings, field ...) and `bigops` for indexed operations.

In this formalization, representations are defined as a subclass of algebras and modules. For the latter we have formalized a theory of linear algebra which covers the theories of finitely generated modules over algebras or fields. This development consists of four libraries. Together, they are about 2800 lines of code. In these libraries, approximately 95 % of lemmas are proved. The sources are available at the following address : `http://www-sop.inria.fr/marelle/Sidi.Biha/reptheo`

The first perspective of this work is the formalization of more advanced results of representation theory. The Artin-Wedderburn theorem is an example of such results. Once this achieved, a formalization of character theory is possible. Another interesting perspective is to link this work with work on representations theory that has already been done in computer algebra system like GAP. We can use GAP to compute the irreducible representations of a given group and import it in Coq. We did some experiments to links the two systems by using XML to encode the data exchanged (a finite group generated in GAP), but work on the external communication interfaces of the two systems is still needed.

References

1. Gonthier, G.: Formal proof - The Four-Color Theorem. Notices of the American Mathematical Society **55**(11) (2008)
2. Avigad, J., Donnelly, K., Gray, D., Raff, P.: A formally verified proof of the prime number theorem. CoRR **abs/cs/0509025** (2005)
3. The Flyspeck project: `http://code.google.com/p/flyspeck/`.
4. The C-CoRN project: `http://c-corn.cs.ru.nl/`.
5. Stephen, W.: The Mathematica Book, Fifth Edition. Wolfram Media (2003)
6. The GAP Group: GAP – Groups, Algorithms, and Programming, Version 4.4.12. (2008)
7. Mathematical Components manifesto: `http://www.msr-inria.inria.fr/Projects/math-components/manifesto`.

8. Feit, W., Thompson, J.G.: Solvability of groups of odd order. Pacific Journal of Mathematics **13**(3) (1963) 775–1029
9. Gonthier, G., Mahboubi, A.: A small scale reflection extension for the Coq system. INRIA Technical report, `http://hal.inria.fr/inria-00258384`.
10. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development, Coq'Art: the Calculus of Inductive Constructions. Springer-Verlag (2004)
11. Coq development team: The Coq Proof Assistant Reference Manual, version 8.2. (2009)
12. Gelbart, S.: An Elementary Introduction to the Langlands Program. Bulletin of the American Mathematical Society **10**(2) (1984)
13. Isaacs, I.M.: Character Theory of Finite Groups. American Mathematical Society (1994)
14. James, G., Liebeck, M.: Representations and Characters of Groups. Cambridge University Press (2001)
15. Webb, P.: Finite Group Representations for the Pure Mathematician. The manuscript of the book is available on the author web page `http://www.math.umn.edu/~webb/RepBook/index.html`
16. Gonthier, G., Mahboubi, A., Rideau, L., Tassi, E., Théry, L.: A modular formalisation of finite group theory. In: Theorem Proving in Higher-Order Logics. Volume 4732 of LNCS. (2007) 86–101
17. Bertot, Y., Gonthier, G., Biha, S.O., Pasca, I.: Canonical big operators. In: Theorem Proving in Higher-Order Logics. Volume 5170 of LNCS. (2008) 86–101
18. Pollack, R.: Dependently Typed Records for Representing Mathematical Structure. In: Theorem Proving in Higher Order Logics, TPHOLs 2000, Springer-Verlag (2000) 462–479
19. Geuvers, H., Pollack, R., Wiedijk, F., Zwanenburg, J.: A constructive algebraic hierarchy in Coq. Journal of Symbolic Computation **34**(4) (2002) 271–286
20. Pottier, L.: User contributions in Coq, Algebra (1999) Available at `http://coq.inria.fr/contribs/Algebra.html`.
21. Barthe, G., Capretta, V., Pons., O.: Setoids in type theory. Journal of Functional Programming **13(2)** (March 2003) 261–293
22. Mizar Mathematical Library: `http://mizar.org/library/`.