

Algebraic structures in Axiom and Isabelle: attempt at a comparison

Clemens Ballarin

Institut für Informatik
Universität Innsbruck
6020 Innsbruck, Austria
<http://www4.in.tum.de/~ballarin>

Abstract. The hierarchic structures of abstract algebra pose challenges to the module systems of both programming and specification languages. We relate two existing module systems that are designed for this purpose: the type system of the computer algebra system Axiom, and the module system of the theorem prover Isabelle.

Abstract algebra has been developed at the beginning of the 20th century and has enabled a high degree of reuse of mathematical, in particular algebraic, knowledge in various branches of mathematics. The same kind of reuse is desirable also in mathematical software, both computer algebra systems and provers.

Various facilities to enable code reuse are known for programming languages. These include polymorphism, and module and class systems, for example. Reuse in abstract algebra is particularly demanding and surpasses what programming languages offer normally. The module system of the computer algebra system Axiom [7] is particularly powerful and was designed to accommodate these needs.

Locales are the theory development modules of the theorem prover Isabelle [9, 10]. They too were developed to provide support for abstract algebra. Locales enable to manage specifications and derived theorems of modules effectively. Initial ideas were drawn from the sectioning concept of Coq [8]. Later extensions integrated concepts from algebraic specification, in particular a language to compose specifications [1], and a facility to declare and maintain derived relations between modules [2] (interpretation).

Although modules for programming languages and provers have rather differing requirements — the former deal with reuse of code, the latter with reuse of theorems — interesting insight can be gained from a comparison of both. In the following an attempt at such a comparison is made, by expressing a small fragment of mathematics in both of the systems. We proceed by presenting a piece of abstract algebra first in the usual mathematical notation and then formally in both Axiom and Isabelle (using locales). Both formalisations are then compared.

1 Some Group Theory

Definition 1. A semi group is a tuple $(G, *)$ where G is a set (the carrier set) and $*$ is a binary associative operation on G .

Definition 2. A monoid is a triple $(G, *, 1)$ where $(G, *)$ is a semi group, and $1 \in G$ such that for all $x \in G$

$$1 * x = x \tag{1}$$

$$x * 1 = x \tag{2}$$

Normally, in order to simplify notation, carrier set and algebraic structure are identified. We do so in the following definitions.

Definition 3. A group G is a semi group with $1 \in G$ and an operation $\text{inv} : G \rightarrow G$ such that for all $x \in G$

$$1 * x = x \tag{3}$$

$$\text{inv}(x) * x = 1 \tag{4}$$

It can be shown that (2) holds also for groups, hence

Theorem 1. Every group is a monoid.

The following algebraic structure, whose definition is taken from Jacobson’s text book on Algebra [6] involves two carrier sets. While not all module systems are able to deal with such structures, they occur frequently in algebra. A more prominent example involving two carrier sets are vector spaces.

Definition 4. A group G is said to act on the set S if there exists a map $\circ : G \times S \rightarrow S$ satisfying for all $x, y \in G$ and $s \in S$

$$1 \circ s = s \tag{5}$$

$$(x * y) \circ s = x \circ (y \circ s) \tag{6}$$

2 Axiom: Categories and Domains

Axiom is, to a large extent, implemented in a functional programming language whose type system is unusually rich. It supports dependent types, and its dynamic type discipline has a distinct object-oriented flavour. *Categories* resemble abstract classes, and *domains* provide implementations. A category is asserted to Axiom through a category constructor — that is, a function returning the signature of that category. Multiple inheritance is supported, and the hierarchy of classes forms a directed acyclic graph.

Likewise, a domain constructor is a function, into a category, returning a domain implementing the category. Hence, which domain belongs to which categories is asserted. Categories and domains are parametric. For instance, the category constructor for vector spaces takes as an argument the coefficient field.

Figure 1 shows category declarations for the algebraic structures from Section 1. Each of these declares a function returning the signature of the algebraic structure, which is an object of type `Category`. The function bodies of these declarations are of the form

import with export.

```

SemiGroup(): Category == SetCategory with
  "*" : (% , %) -> %

Monoid(): Category == SemiGroup with
  1 : -> %

Group(): Category == Monoid with
  inv : % -> %

Action(G: Group): Category == SetCategory with
  "o" : (G, %) -> %

```

Fig. 1. Group category declarations in Axiom

The declarations closely follow Section 1, with the exception that Theorem 1 has been incorporated in the definition of `Group`. Its import is `Monoid` rather than `SemiGroup`. The `%` sign denotes the carrier type. In `Category Action` the group and its carrier is introduced as an argument of the category constructor.

3 Locales: Morphisms and Interpretation

Locales are an extra-logical concept built on top of the meta-logic of Isabelle, which is available for all object-logics of Isabelle, including higher-order logic and ZF set theory. A locale contains a specification and can be seen as a storage container for theorems implied by that specification. Theorems may be added to a locale at any time, not just during the creation of the locale.

Locales are parametric. The parameters, which occur in both specification and theorems, can be replaced by terms, thus creating instances of the locale. This mapping from parameters to terms is called a *morphism*. By a (simple) property of the meta-logic images of locales under morphisms are consistent. This means that the instantiated specification implies the instantiated theorems.

Locales are, like to Axiom's categories, hierarchic. A locale may import one or several other locales, with the effect that the parameters and specifications of the imported locales become part of the importing locale. The same applies to theorems. Import may also be through morphisms. Then, instantiated specification and instantiated theorems become part of the locale.

Like with Axiom's categories, the import hierarchy of locales is a directed acyclic graph, where additionally, edges are labelled with morphisms. Besides of the import hierarchy, *interpretation* relations between locales are maintained. If the specification of one locale implies (an instance of) the specification of another locale, the corresponding instances of the theorems of that locale are also theorems of the locale. Locales provide an interpretation command through which interpretation relations between locales can be stated. Unlike import, which is declared, interpretation is a consequence of the specification of the involved locales and must be accompanied by a proof.

```

locale semigroup =
  fixes mult :: "'a => 'a => 'a" (infixl "*" 60)
  assumes assoc: "(x * y) * z = x * (y * z)"

locale monoid = semigroup +
  fixes one :: "'a" ("1")
  assumes lone: "1 * x = x"
  and rone: "x * 1 = x"

locale group = semigroup +
  fixes one :: "'a" ("1")
  and inv :: "'a => 'a"
  assumes lone: "1 * x = x"
  and linv: "inv x * x = 1"

interpretation group < monoid

locale action = group +
  fixes op :: "'a => 'b => 'b" (infixl "o" 60)
  assumes op_one: "1 o s = s"
  and op_assoc: "(x * y) o s = x o (y o s)"

```

Fig. 2. Group locale declarations in Isabelle

The management of hierarchic information in locales is similar to development graphs [5], and indeed they can be phrased in terms of development graphs [2, 3].

Figure 2 shows locale declarations corresponding to Section 1. Its structure is similar to the category declarations from Figure 1 with the exception that `group` directly extends `semigroup`. Instead, Theorem 1 is present in the declarations, as an interpretation¹ making theorems a user may add to `monoid` available in `group`. Carrier sets are represented by types, and are denoted by the type variables `'a` and `'b`.

4 Comparison

A comparison between the module system of a programming language and that of a theorem prover may appear an unfeasible endeavour. But both systems are designed to reflect hierarchies of abstract algebraic structures, and this is a natural starting point for a comparison. Axiom's categories correspond to abstract algebraic structures, and so do locales. The following comparison is restricted to aspects related to the representation of hierarchies of algebraic structures.

Programming vs. specification language. Since Axiom is a programming language, a category only represents the signature, while locales exhibit both signature and specification.²

¹ In Figure 2 the proof justifying the interpretation is omitted.

² An extension of Axiom by specifications was presented by Poll and Thompson [11].

Role of parameters. The parameters of a category are the arguments of the category. They are distinct from the signature, which is its result. This distinction is not present in locales: here the parameters *are* the signature, and the locale can be seen as a relation between the parameters. Hence, while in Axiom the category constructor for action groups takes the group signature as an argument and returns the signature of the set structure the group acts upon, the corresponding locale has parameters for both the operations of the group and the set. We may call Axiom's categories *functorial* and Isabelle's locales *relational* representations of algebraic structures.

Morphisms. Morphisms are not present in Axiom and they are not fully developed in locales.³ Figure 2 contains no explicit use of morphisms. An example of their use is the derivation of an additive group from a multiplicative one.

```
locale additive_group =  
  group add (infixl "+" 55) zero ("0") uminus ("- _")
```

The parameters, which have a canonical positional order, are renamed and receive new syntax. Morphisms are convenient for the construction of rings, or when defining the notion of a group homomorphism. In order to achieve such structures in Axiom, two separate hierarchies are implemented.

Interpretation. Categories only model the declared import hierarchy between abstract structures. Derived dependencies (interpretations) cannot be asserted later, while with locales this is possible. Interpretation is a standard device in a specification language, since it permits to establish relations between modules that were not anticipated when these modules were declared. Interpretation does not make sense in the context of a programming language, since there are no specifications and consequently no relations between them can be proved. Nevertheless, a facility to declare module dependencies not anticipated by a library designer might be a useful asset in a programming language. The algebraic hierarchy implemented in Axiom has been criticised as hard to extend. This might be an indication that similar means are missing in Axiom's module system.

5 Conclusions and Directions for Further Research

We have presented and compared aspects of two module systems for the representation of abstract algebraic structures. Both are based on strong foundations: Axiom's programming language has dependent types, which is unusual for functional languages. The hierarchic structure of locales can be described with development graphs. This is combined with Isabelle's meta-logic, which is a variant of higher-order logic. In contrast, specification languages are normally based on variants of first-order logic.

The comparison is preliminary and has highlighted interesting differences. Some of these differences are rooted in the fact that we have compared a programming and a specification language: Axiom's categories are functorial, while Isabelle's locales resemble relations with additional infrastructure for the management of hierarchies.

³ The current implementation is restricted to parameter renamings, not arbitrary instances.

In the author's view this is the fundamental difference between Axiom's type system and locales. Once a concise characterisation has been obtained by abstracting away from technical aspects of both systems, two goals may be addressed:

- Compare the notions of functorial vs. relational approach to specifications with respect to their strengths.
- Classify other module systems, like the one for Coq [4], or FoCal [12].

Additionally, concepts from specification languages might be fruitfully carried over to programming languages. While interpretation *per se* is only meaningful in the presence of specifications, a similar mechanism to modify an existing module hierarchy might also be useful in programming languages. Similarly, morphisms might enable more complex import operations than inheritance.

References

1. C. Ballarin. Locales and locale expressions in Isabelle/Isar. In S. Berardi, M. Coppo, and F. Damiani, editors, *Types for Proofs and Programs, TYPES 2003, Torino, Italy*, LNCS 3085, pages 34–50. Springer, 2004.
2. C. Ballarin. Interpretation of locales in Isabelle: Managing dependencies between locales. Technical Report TUM-I0607, Technische Universität München, 2006.
3. C. Ballarin. Interpretation of locales in Isabelle: Theories and proof contexts. In J. M. Borwein and W. M. Farmer, editors, *Mathematical knowledge management, MKM 2006, Wokingham, UK*, LNCS 4108, pages 31–43. Springer, 2006.
4. J. Chrzaszcz. Implementing modules in the Coq system. In D. Basin and B. Wolff, editors, *Theorem proving in higher order logics: TPHOLs 2003, Rome, Italy*, LNCS 2758, pages 270–286. Springer, 2003.
5. D. Hutter. Management of change in structured verification. In *Automated Software Engineering, ASE 2000, Grenoble, France*, pages 23–31. IEEE Computer Society, 2000.
6. N. Jacobson. *Basic Algebra*, volume I. Freeman, 2nd edition, 1985.
7. R. D. Jenks and R. S. Sutor. *AXIOM. The scientific computation system*. Numerical Algorithms Group, Ltd. and Springer-Verlag, Oxford and New York, 1992.
8. F. Kammüller, M. Wenzel, and L. C. Paulson. Locales: A sectioning concept for Isabelle. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *Theorem Proving in Higher Order Logics: TPHOLs'99, Nice, France*, LNCS 1690, pages 149–165. Springer, 1999.
9. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. LNCS 2283. Springer, 2002.
10. L. C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5(3):363–397, 1989.
11. E. Poll and S. Thompson. Adding the axioms to Axiom: Towards a system of automated reasoning in Aldor. Technical Report 6–98, Computing Laboratory, University of Kent, May 1998. Presented at the workshop Calculemus and Types, Eindhoven, Netherlands, July 1998.
12. V. Prevosto. Certified mathematical hierarchies: the FoCal system. In T. Coquand, H. Lombardi, and M.-F. Roy, editors, *Mathematics, Algorithms, Proofs*, number 05021 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005.