AN ACL2 FORMALIZATION OF ALGEBRAIC STRUCTURES

JÓNATHAN HERAS, FRANCISCO-JESÚS MARTÍN-MATEOS, AND VICO PASCUAL

Abstract. In this paper we present a hierarchy of algebraic structures developed in the ACL2 Theorem Prover. This question had not been undertaken in this system up to now. A methodology and a bunch of tools to handle the structures and morphisms included in our hierarchy have been designed. In order to ensure the applicability of our methodology and tools, a major issue when developing formal proofs, we have proved some results coming from Universal Algebra.

Introduction

Proof Assistant tools are nowadays mature enough to tackle non-trivial mathematical problems; we can cite, among others, the formalizations of the Four Color Theorem [5], the Fundamental Theorem of algebra [4] or the Kepler conjecture [13].

The choice of a convenient representation for algebraic structures has been a cornerstone of all these projects. As a result of that, several algebraic hierarchies have been developed in Proof Assistant tools like Coq, Isabelle or Hol. However, as far as we know, this question had not been undertaken in ACL2 [10], a theorem prover designed to verify properties of code written in the programming language Common Lisp [6].

In this paper we present an ACL2 algebraic hierarchy which allows one to create theories about usual mathematical structures (the hierarchy ranges from setoids to R-modules including structures such as groups or rings) and morphisms between those structures. This ACL2 hierarchy is not only an end, but also a means to achieve our final goal of verifying actual code of the Kenzo Computer Algebra system [3]. To test the suitability of our framework, we present how to use it to prove a result from Universal Algebra.

1. An algebraic hierarchy in ACL2

The hierarchy of mathematical structures and morphisms depicted in Figure 1 has been developed in ACL2, a preliminary approach was presented in [7]. In the left side of Figure 1, there are the mathematical structures of our hierarchy, ranging from *setoids* to *R-modules*. A detailed description of each one of these structures can be seen in [2].

A continuous arrow with an open triangle represents an *inheritance* relationship modeling that the source mathematical structure *is-a* target mathematical structure. Whereas a continuous arrow with a normal tip describes a *use* relationship in the sense that the target mathematical structure *is used* to define the source one.

The morphisms included in our hierarchy are presented in the right side of Figure 1. It is worth noting that a morphism always consists of a source structure, A of type T, a target

Partially supported by Ministerio de Educación y Ciencia, project MTM2009-13842-C02-01, and by the European Union's 7th Framework Programme under grant agreement nr. 243847 (ForMath).

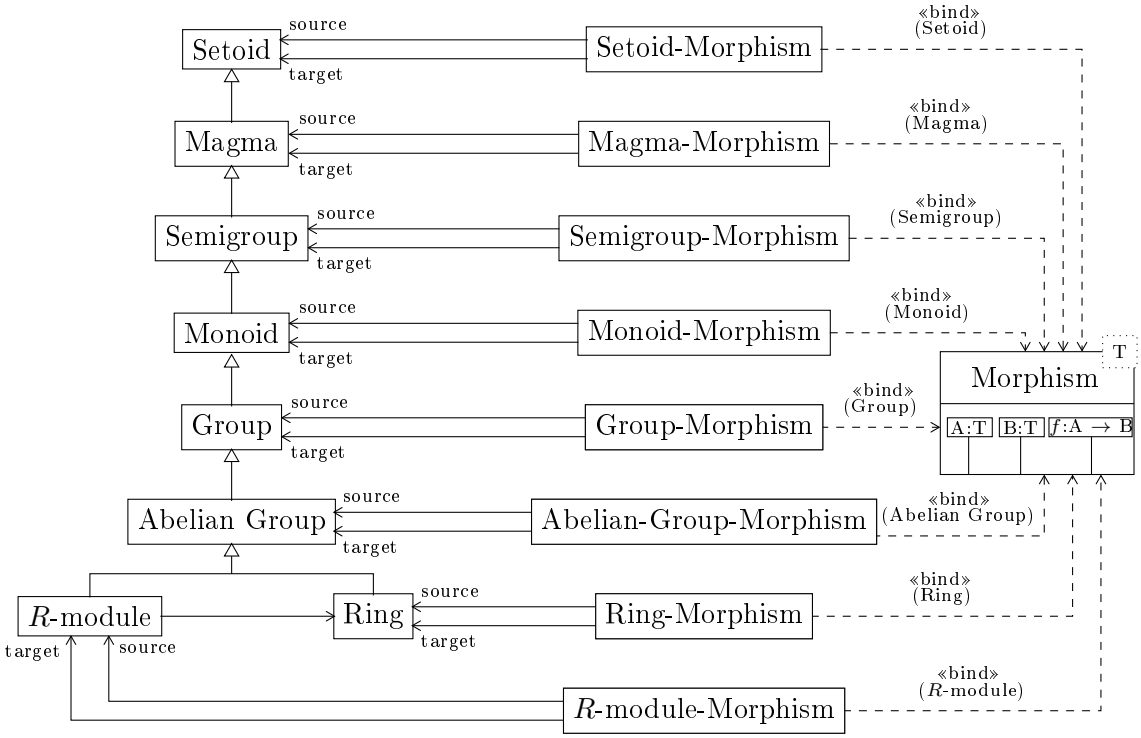

FIGURE 1. Hierarchy of mathematical structures and morphisms

structure, B of type T, and a map $f : A \to B$. Therefore, a morphism can be seen as a *template*, using UML terminology, which is parameterized by a mathematical structure T. An actual morphism is produced by binding the parameter T to one of the concrete mathematical structures of our hierarchy. This is the meaning of the dashed arrows in our diagram. As a final remark, the two continuous arrows between a T-morphism and the T structure describe, as in the case of mathematical structures, a *use* relationship with *role*. Namely, one T-structure plays the *role* of the *source* in the definition of the T-morphism and the other one the *role* of the *target*.

2. Application: Universal Algebra

We have defined several tools to make the use of the mathematical structures and morphisms of our hierarchy easier in ACL2. Namely, for every element of our hierarchy, our tools allow us to: *represent* it, *certify* that an object fulfills the definitional axioms which characterize it, and *generate generic theories* about it.

Let us present their use by means of a result coming from Universal Algebra [2], the *Subalgebra criterion*. This criterion says that given $\mathcal{X} = (X, op_1, \ldots, op_n)$, a T-structure where X is the underlying set of $\mathcal{X}$ and $op_1, \ldots, op_n$ are the operations defined on X; and Y, a subset of X closed with respect to $op_1, \ldots, op_n$; then $\mathcal{Y} = (Y, op_1, \ldots, op_n)$ is also a T-structure. This result has been proved for all the structures of our hierarchy, let us present the proof for the simplest one: *setoids*.

A *setoid* $\mathcal{X} = (X, \sim_X)$ is a set X together with an equivalence relation $\sim_X$ on it. Then, a setoid can be represented by means of two functions: the characteristic function of the underlying set, the *invariant*, and a binary function encoding the *equivalence relation*. This is carried out in ACL2 by means of a record, implemented with the `defstructure` macro [1], with two fields (`inv` and `eq`) which store respectively the names of the invariant function and the intended equivalence relation (in addition, these functions must have been introduced previously). The following `defstructure` construction is used to define the setoid structure.

```
(defstructure setoid inv eq)
```

Now, we can create concrete setoids using this representation; for instance, the setoid whose underlying set is the set of integer numbers (this invariant function is encoded in ACL2 with the `integerp` function) having the same absolute value (encoded with `eq-abs`) can be assigned to an ACL2 constant, called `*Zabs*`, as follows.

```
(defconst *Zabs* (make-setoid :inv 'integerp :eq 'eq-abs))
```

As the Subalgebra criterion is a universal property; the definition of a generic setoid is necessary. In the case of setoids this task is carried out by means of a macro called `defgeneric-setoid`. This macro takes as argument a symbol, for instance `X`, and produces, on the one hand, the constant `*X*` which stores a generic setoid (whose components are `X-inv`, the invariant function, and `X-eq`, the intended equivalence relation); and, on the other hand, the theorem `*X*-is-a-setoid` which ensures that `*X*` satisfies the setoid axioms.

From this generic setoid, we can define a *generic subset* Y of X using the `encapsulate` principle [9], an ACL2 mechanism to introduce functions only assuming some properties about them.

```
(encapsulate
   (((Y-inv *) => *))

   (defthm Y-subset-of-X
      (implies (Y-inv x) (X-inv x)))
)
```

Now, we can build a `setoid` instance where `Y-inv` is the invariant function and `X-eq` is the equivalence relation; and store it in the constant `*Y*`.

```
(defconst *Y* (make-setoid :inv 'Y-inv :eq 'X-eq))
```

Eventually, we can certify that `*Y*` is really a setoid using the tool defined for this purpose, called `check-setoid-p`.

```
(check-setoid-p *Y*)
```

The above macro expands into a call of `defthm` whose name is `*Y*-is-a-setoid`. This event states the setoid definitional axioms for the components of `*Y*`. In this way, we have proved the Subalgebra criterion for setoids; and this result can be instantiated for concrete setoids as the one defined previously.

It is worth noting that, in this case, ACL2 is able to prove the theorem `*Y*-is-a-setoid` without any external help; but, in several cases (as usual in ACL2), the user must provide some guidance to the system introducing some auxiliary lemmas.

3. Conclusions and Further work

We have presented the first hierarchy of algebraic structures in ACL2. The different tools that we have devised facilitate the development of theories about the structures and morphisms of the hierarchy. Moreover, the hierarchy is flexible enough to be extended without any special hindrance.

In the future, we would like to use the present work as a basis to tackle complex formalizations. Namely, an appealing problem is the development of an ACL2 fully *certified* version of a portion of the Kenzo Computer Algebra system, a successful tool devoted to Algebraic Topology written in the same language in which ACL2 is built on. In particular, that new formally verified Computer Algebra system should include the minimal Kenzo functionality to compute simplicial homology groups. Some first steps towards this goal have been given, see [8, 12, 11], but much more work is still needed.

References

[1] B. Brock. `defstructure` for ACL2 version 2.0. Technical report, Computational Logic, Inc., 1997.
[2] K. Denecke and S. L. Wismath. *Universal Algebra and Applications in Theoretical Computer Science*. Chapman Hall/CRC, 2002.
[3] X. Dousson, J. Rubio, F. Sergeraert, and Y. Siret. *The Kenzo program*. Institut Fourier, Grenoble, 1998. `http://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/`.
[4] H. Geuvers, F. Wiedijk, J. Zwanenburg, R. Pollack, and H. Barendregt. *The Fundamental Theorem of Algebra project*. `http://www.cs.ru.nl/~freek/fta/`.
[5] G. Gonthier. *Formal proof: The four-color theorem*. Notices of the AMS, 55(11):1382–1393, 2008.
[6] P. Graham. *ANSI Common Lisp*. Prentice Hall, 1996.
[7] J. Heras. *Mathematical Knowledge Management in Algebraic Topology*. PhD thesis, 2011.
[8] J. Heras, V. Pascual, and J. Rubio. *Proving with ACL2 the correctness of simplicial sets in the Kenzo system*. In Proceedings International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR'2010), volume 6564 of Lectures Notes in Computer Science, pages 37–51, 2011.
[9] M. Kaufmann and J S. Moore. *Structured Theory Development for a Mechanized Logic*. Journal of Automated Reasoning, 26(2):161–203, 2001.
[10] M. Kaufmann and J S. Moore. *ACL2 version 4.3*, 2011. `http://www.cs.utexas.edu/users/moore/acl2/`.
[11] L. Lambán, F. J. Martín-Mateos, J. Rubio, and J. L. Ruíz-Reina. *Formalization of a normalization theorem in simplicial topology*. To be published in Annals of Mathematics and Artificial Intelligence.
[12] L. Lambán, F. J. Martín-Mateos, J. Rubio, and J. L. Ruiz-Reina. *Applying ACL2 to the Formalization of Algebraic Topology: Simplicial Polynomials*. In Proceedings Interactive Theorem Proving (ITP'2011), volume 6898 of Lecture Notes in Computer Science, pages 200–215, 2011.
[13] S. Obua and T. Nipkow. *Flyspeck II: The basic linear programs*. Annals of Mathematics and Artificial Intelligence, 56:245–272, 2009.

Departamento de Matemáticas y Computación. Universidad de La Rioja.
E-mail address: `{jonathan.heras,vico.pascual}@unirioja.es`

Grupo de Lógica Computacional, Departamento de Ciencias de la Computación e Inteligencia Artificial. Universidad de Sevilla
E-mail address: `fjesus@us.es`