An ACL2 formalization of Algebraic structures*

J. Heras[1], F.J. Martín-Mateos[2] and V. Pascual[1]

[1]Departmento de Matemáticas y Computación, Universidad de La Rioja
[2]Grupo de Lógica Computacional, Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Sevilla

XIII Encuentro de Álgebra Computacional y Aplicaciones (EACA 2012)

*Partially supported by Ministerio de Educación y Ciencia, project MTM2009-13842-C02-01, and by European Commission FP7, STREP project ForMath, n. 243847

Table of Contents

Table of Contents

Interactive Proof Assistants

- What is an Interactive Proof Assistant?
  - Software tool for the development of formal proofs
  - Man-Machine collaboration:
    - Human: design the proofs
    - Machine: fill the gaps
  - Examples: Isabelle, Hol, ACL2, COQ, ...

Interactive Proof Assistants

- What is an Interactive Proof Assistant?
  - Software tool for the development of formal proofs
  - Man-Machine collaboration:
    - Human: design the proofs
    - Machine: fill the gaps
  - Examples: Isabelle, Hol, ACL2, Coq, . . .
- Applications:
  - Mathematical proofs:
    - Four Color Theorem
    - Fundamental Theorem of Algebra
    - Kepler conjecture
  - Software and Hardware verification:
    - C compiler
    - AMD5K86 microprocessor
    - . . .

Algebraic structures in theorem provers

Foundation for large proof developments

Algebraic structures in theorem provers

Foundation for large proof developments

- Coq:
  - CCorn hierarchy
  - SSReflect hierarchy
  - ...
- Isabelle
- Nuprl
- Lego

ACL2

- ACL2 (A Computational Logic for an Applicative Common Lisp)

ACL2

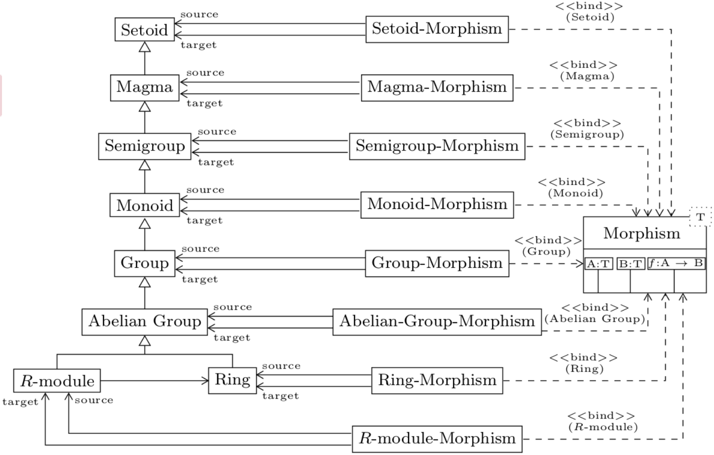
- ACL2 (A Computational Logic for an Applicative Common Lisp)
- ACL2:
    - Programming Language
    - First-Order Logic
    - Theorem Prover

ACL2

- ACL2 (A Computational Logic for an Applicative Common Lisp)
- ACL2:
  - Programming Language
  - First-Order Logic
  - Theorem Prover
- Proof techniques:
  - Simplification
  - Induction
  - The Method

Table of Contents

Hierarchy of mathematical structures and morphisms



Modeling setoids

A *setoid* $\mathcal{X} = (X, \sim_X)$ is a set X together with an equivalence relation $\sim_X$ on it

Modeling setoids

A *setoid* $\mathcal{X} = (X, \sim_X)$ is a set X together with an equivalence relation $\sim_X$ on it

Example

Setoid whose underlying set is the set of integer numbers having the same absolute value

```
integerp
```

```
(defun eq-abs (a b)
  (equal (abs a) (abs b)))
```

Modeling setoids

A *setoid* $\mathcal{X} = (X, \sim_X)$ is a set X together with an equivalence relation $\sim_X$ on it

Example

Setoid whose underlying set is the set of integer numbers having the same absolute value

```
integerp

(defun eq-abs (a b)
  (equal (abs a) (abs b)))

(implies (integerp x) ;; REFLEXIVE
         (eq-abs x x))

(implies (and (integerp x) (integerp y) (eq-abs x y)) ;; SYMMETRY
         (eq-abs y x))

(implies (and (integerp x) (integerp y) (integerp z) ;; TRANSITIVE
              (eq-abs x y) (eq-abs y z))
         (eq-abs x z))
```

Modeling setoids

```
(encapsulate
   ;  SIGNATURES
   (((X-inv *) => *)
    ((X-eq * *) => *))

   ;  ASSUMPTIONS
   (defthm X-reflexive
      (implies (X-inv x)
               (X-eq x x)))

   (defthm X-symmetry
      (implies (and (X-inv x) (X-inv y) (X-eq x y))
               (X-eq y x)))

   (defthm X-transitive
      (implies (and (X-inv x) (X-inv y) (X-inv z) (X-eq x y) (X-eq y z))
               (X-eq x z)))
)
```

Modeling setoids

```
(encapsulate
   ;  SIGNATURES
   (((X-inv *) => *)
    ((X-eq * *) => *))

   ;  ASSUMPTIONS
   (defthm X-reflexive
      (implies (X-inv x)
               (X-eq x x)))

   (defthm X-symmetry
      (implies (and (X-inv x) (X-inv y) (X-eq x y))
               (X-eq y x)))

   (defthm X-transitive
      (implies (and (X-inv x) (X-inv y) (X-inv z) (X-eq x y) (X-eq y z))
               (X-eq x z)))
)

(defthm symmetry-transitive
   (implies (and (X-inv x) (X-inv y) (X-inv z) (X-eq y x) (X-eq y z))
            (X-eq x z)))
```

Modeling setoids

Enhancements:

- Structure gathering functional components
- Definition of macros to certify definitional axioms
- Macro to define generic instances

```
(defstructure setoid
  inv eq)

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

(check-setoid-p *S*)

(defgeneric-setoid X)
```

Modeling setoids

Enhancements:

- Structure gathering functional components
- Definition of macros to certify definitional axioms
- Macro to define generic instances

```
(defstructure setoid
  inv eq)

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

(check-setoid-p *S*)

(defgeneric-setoid X)
```

Modeling setoids

Enhancements:

- Structure gathering functional components
- Definition of macros to certify definitional axioms
- Macro to define generic instances

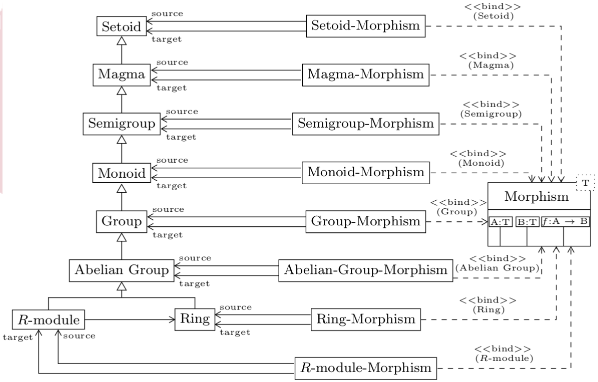
```
(defstructure setoid
  inv eq)

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

(check-setoid-p *S*)

(defgeneric-setoid X)
```

A hierarchy of structures and morphisms



- Records to represent them
- Definition of macros to certify definitional axioms
- Macros to define generic instances

A hierarchy of structures and morphisms

A magma is a setoid with a closed and compatible binary operation

```
(defstructure magma
  setoid binary-op)
```

A hierarchy of structures and morphisms

A magma is a setoid with a closed and compatible binary operation

```
(defstructure magma
  setoid binary-op)
```

```
(defconst *Zmagma* (make-magma
                         :setoid (make-setoid :inv 'integerp :eq 'eq-abs)
                         :binary-op '+))
```

A hierarchy of structures and morphisms

A magma is a setoid with a closed and compatible binary operation

```
(defstructure magma
  setoid binary-op)
```

```
(defconst *Zmagma* (make-magma
                        :setoid (make-setoid :inv 'integerp :eq 'eq-abs)
                        :binary-op '+))
```

$$\texttt{check-magma-p} = \begin{cases} \texttt{check-setoid-p} \\ \texttt{binary-op} \text{ is closed} \\ \texttt{binary-op} \text{ is compatible} \end{cases}$$

Table of Contents

Homology groups

Definition

Let $f : G_1 \to G_2$ and $g : G_2 \to G_3$ be abelian group morphisms such that $\forall x \in G_1, gf(x) \sim_{G_3} 0_{G_3}$ (where 0_{G_3} is the neutral element of G_3), then the homology group of (f, g), denoted by $H_{(f,g)}$, is the abelian group $H_{(f,g)} = ker(g)/im(f)$.

Homology groups

Definition

Let $f : G_1 \to G_2$ and $g : G_2 \to G_3$ be abelian group morphisms such that $\forall x \in G_1, gf(x) \sim_{G_3} 0_{G_3}$ (where 0_{G_3} is the neutral element of G_3), then the homology group of (f,g), denoted by $H_{(f,g)}$, is the abelian group $H_{(f,g)} = ker(g)/im(f)$.

Goal

Use our framework to define $H_{(f,g)}$ and prove that it is an abelian group

Homology groups

Definition of 3 generic abelian groups:

```
(defgeneric-abelian-group G1)
(defgeneric-abelian-group G2)
(defgeneric-abelian-group G3)
```

Components of these groups:

- G<i>-inv: the underlying set
- G<i>-eq: the equivalence relation
- G<i>-binary-op: the binary operation
- G<i>-id-elem: the neutral element
- G<i>-inverse: the inverse operator

Homology groups

Definition of two generic abelian group morphisms satisfying nilpotency:

```
(encapsulate
   ;  SIGNATURES
   (((f *) => *)
    ((g *) => *))

   ;  GENERIC ABELIAN GROUP MORPHISMS DEFINITION
   (defconst *f*
     (make-abelian-group-morphism :source *G1* :target *G2* :map 'f))
   (defconst *g*
     (make-abelian-group-morphism :source *G2* :target *G3* :map 'g))

   ;  ABELIAN GROUP MORPHISM AXIOMS
   (check-abelian-group-morphism-p *f*)
   (check-abelian-group-morphism-p *g*)

   ;  NILPOTENCY CONDITION
   (defthm nilpotency-condition
      (implies (G1-inv x)
               (G3-eq (g (f x)) (G3-id-elem))))
)
```

Homology groups

$ker(g) = \{x \in G_2 : g(x) \sim_{G_3} 0_{G_3}\}$

```
(defun ker-g-inv (x)
   (and (G2-inv x)
        (G3-eq (g x) (G3-id-elem))))

(defconst *ker-g*
   (make-abelian-group :group
        (make-group :monoid
             (make-monoid :semigroup
                  (make-semigroup :magma
                       (make-magma :setoid
                            (make-setoid :inv 'ker-g-inv :eq 'G2-eq)
                                   :binary-op 'G2-binary-op))
                          :id-elem 'G2-id-elem)
                     :inverse 'G2-inverse))
```

Homology groups

$$im(f) = \{x \in G_2 : \exists y \in G_1, f(y) \sim_{G_2} x\}$$

```
(defun-sk im-f-inv (x)
  (exists (y)
          (and (G2-inv x) (G1-inv y) (G2-eq (f y) x))))

(defconst *im-f*
   (make-abelian-group :group
        (make-group :monoid
             (make-monoid :semigroup
                  (make-semigroup :magma
                       (make-magma :setoid
                            (make-setoid :inv 'im-f-inv :eq 'G2-eq)
                                   :binary-op 'G2-binary-op))
                         :id-elem 'G2-id-elem)
                   :inverse 'G2-inverse))
```

Homology groups

$H_{(f,g)} = ker(g)/im(f)$

Equivalence Relation: $\forall x, y \in ker(g), x \sim_{im(f)} y \Leftrightarrow xy^{-1} \in im(f)$

```
(defun im-f-eq (x y)
  (im-f-inv (G2-binary-op x (G2-inverse y))))
```

```
(defconst *homology-fg*
  (make-abelian-group :group
       (make-group :monoid
            (make-monoid :semigroup
                 (make-semigroup :magma
                      (make-magma :setoid
                           (make-setoid :inv 'ker-g-inv :eq 'im-f-eq)
                                  :binary-op 'G2-binary-op))
                         :id-elem 'G2-id-elem)
                   :inverse 'G2-inverse))
```

```
(check-abelian-group-p *homology-fg*)
```

Table of Contents

Benefits of our methodology

- Application to Algebraic Topology

Benefits of our methodology

- Application to Algebraic Topology

Definition

A chain complex is a pair $(C_n, d_n)_{n\in\mathbb{Z}}$ where $(C_n)_{n\in\mathbb{Z}}$ is a graded R-module indexed on the integers and $(d_n)_{n\in\mathbb{Z}}$ (the differential map) is a graded R-module endomorphism of degree -1 ($d_n : C_n \rightarrow C_{n-1}$) such that $d_{n-1}d_n = 0$ (this property is known as nilpotency condition).

Let $(C_n, d_n)_{n\in\mathbb{Z}}$ and $(D_n, \widehat{d}_n)_{n\in\mathbb{Z}}$ be two chain complexes, a chain complex morphism between them is a family of R-module morphism $(f_n)_{n\in\mathbb{Z}}$ such that $\widehat{d}_n f_n = f_{n-1} d_n$ for each $n \in \mathbb{Z}$.

Benefits of our methodology

- Application to Algebraic Topology

Definition

A chain complex is a pair $(C_n, d_n)_{n\in\mathbb{Z}}$ where $(C_n)_{n\in\mathbb{Z}}$ is a graded R-module indexed on the integers and $(d_n)_{n\in\mathbb{Z}}$ (the differential map) is a graded R-module endomorphism of degree -1 ($d_n : C_n \to C_{n-1}$) such that $d_{n-1}d_n = 0$ (this property is known as nilpotency condition).
Let $(C_n, d_n)_{n\in\mathbb{Z}}$ and $(D_n, \widehat{d}_n)_{n\in\mathbb{Z}}$ be two chain complexes, a chain complex morphism between them is a family of R-module morphism $(f_n)_{n\in\mathbb{Z}}$ such that $\widehat{d}_n f_n = f_{n-1}d_n$ for each $n \in \mathbb{Z}$.

Definition

Let $C_* = (C_n, dC_n)_{n\in\mathbb{Z}}$ and $D_* = (D_n, dD_n)_{n\in\mathbb{Z}}$ be two chain complexes and $\phi : D_* \to C_*$ be a chain complex morphism. Then the cone of ϕ, denoted by $Cone(\phi) = (A_n, dA_n)_{n\in\mathbb{Z}}$, is defined as: $A_n := C_{n+1} \oplus D_n$; and

$$dA_n := \begin{bmatrix} dC_{n+1} & \phi \\ 0 & -dD_n \end{bmatrix}$$

Benefits of our methodology

	Definition of generic chain complex morphism	Definition of cone construction	Proof of the correctness of the construction
from scratch	19 function symbols 19 witnesses 84 axioms	9 definitions	49 theorems 34 auxiliary lemmas
hierarchical	1 macro call	9 definitions 1 chain-complex	1 macro call 34 auxiliary lemmas

Table of Contents

Conclusions and Further work

Conclusions:

- ACL2 infrastructure to deal with algebraic structures and morphisms
- Methodology to handle other mathematical structures
- Benefits when proving

Conclusions and Further work

Conclusions:

- ACL2 infrastructure to deal with algebraic structures and morphisms
- Methodology to handle other mathematical structures
- Benefits when proving

Further work:

- Apply methodology to other structures
- Formalizing the generic theory of Universal Algebra
- Automatic generation of tools for morphisms
- Certification of critical fragments of Kenzo

An ACL2 formalization of Algebraic structures

J. Heras[1], F.J. Martín-Mateos[2] and V. Pascual[1]

[1]Departmento de Matemáticas y Computación, Universidad de La Rioja
[2]Grupo de Lógica Computacional, Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Sevilla

XIII Encuentro de Álgebra Computacional y Aplicaciones
(EACA 2012)