Automated theorem proving in Simplicial Topology with ACL2

Mirian Andrés

University of La Rioja (U.R.), Spain

ICTP, Trieste (Italy)
2008 August

AGENDA

- Introduction
- Kenzo
- ACL2
- Our main proposal
- A concrete proposal
- An example
  - Simplicial Topology in ACL2
  - A developed example
  - A direct proof
  - A proof based on abstract reduction systems
- Conclusions and further work

Introduction

- Research group directed by ***Julio Rubio***
- Different lines with different people working in them
- My line: automatic theorem provers

Kenzo

The Common Lisp system
Kenzo
to compute in Algebraic Topology

- tested but … not always

- its programs correctness has not been proved!

- we are concentrated now in **increasing its reliability**

- A formal approach to increase our confidence in the correctness of a computer program: ***verification***

Use mathematical methods to prove that the program meets its intended specification

Formal verification of programs

*Instead of debugging a **program**, one should prove that it meets its **specifications**, and this **proof** should be **checked by a computer program***

(John McCarthy, “A Basis for a Mathematical Theory of Computation” 1961)

- What do we need to formally verify a program?
  - **A programming language**
  - **A logic**
  - **A theorem prover**

The ACL2 system

- ACL2 stands for “**A** **C**omputational **L**ogic for an **A**pplicative **C**ommon **L**isp”

- Developed in the University of Texas at Austin by J Moore and Matt Kaufmann, since 1994

- Its predecessor is Nqthm, also (well) known as the Boyer-Moore theorem prover

- Successfully used in the industry: hardware verification

- But also used in the verification of software and in formalization of mathematics

Our main proposal

Our idea: using ACL2 to verify the actual Kenzo programs

But ... Kenzo uses higher order functional programming

- mechanized proofs in Isabelle for some theoretical algorithms used in **Kenzo** (*Aransay's proof in Isabelle of the Basic Perturbation Lemma*)

- distance from the Kenzo code to the theories and proofs in Isabelle

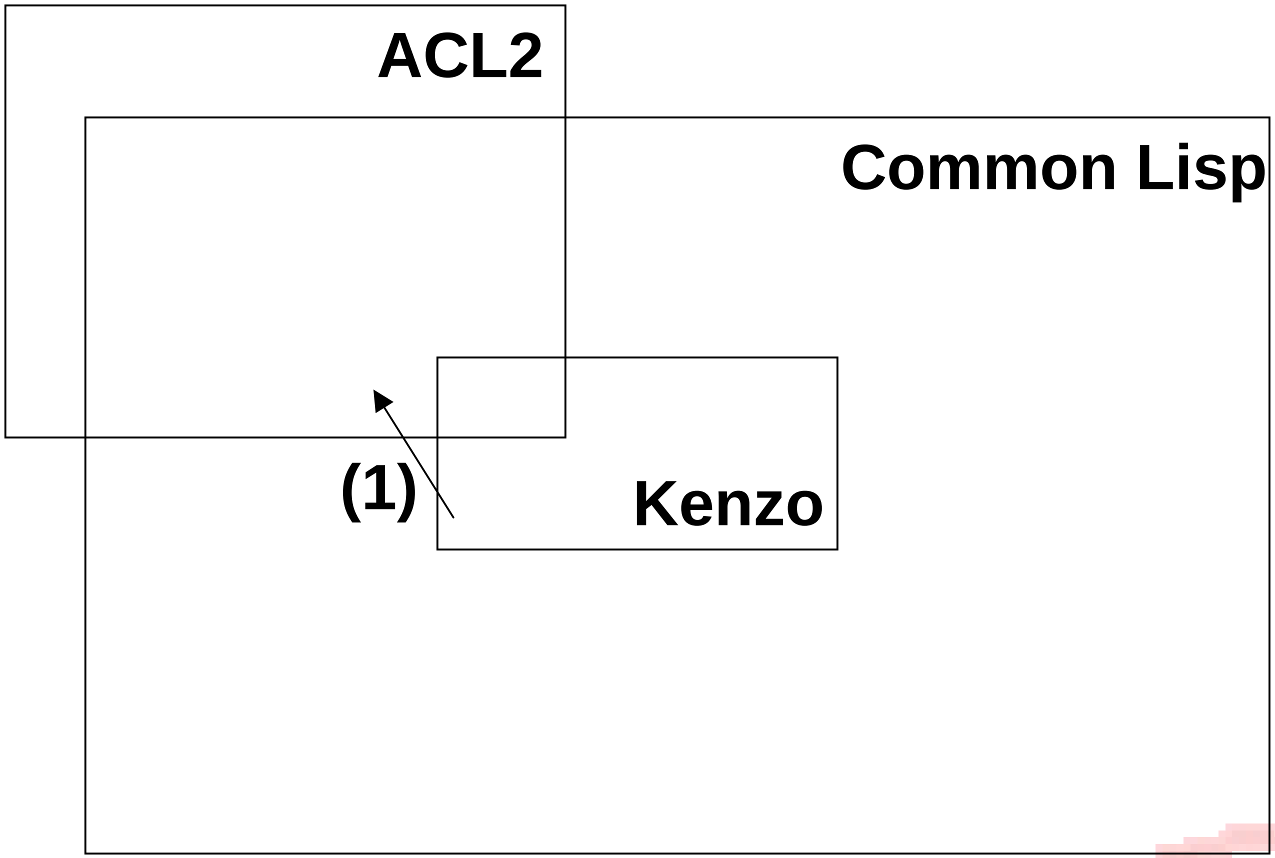
How could we increase the reliability of Kenzo with ACL2?

Our proposal:

Choose, reprogram and verify in ACL2 first-order fragments of Kenzo related with Simplicial Topology

ACL2: A Computational Logic for Applicative Common Lisp

ACL2 is an extension of a part of Common Lisp



program1 —(1)→ **program2**

program1 is

- already written
- Common Lisp (not ACL2)
- efficient
- tested
- unproved

program2 is

- specially designed to be proved
- ACL2 (and Common Lisp)
- efficient or not : irrelevant
- tested
- proved in ACL2

`program2`

"is ***supposed*** to be equivalent to"

`program1`

we do not expect to ***prove*** this equivalence

but to use it to do ***automated testing***

```
(defun automated-testing ()
  (let ((case (generate-test-case)))
    (if (not (equal (program1 case)
                    (program2 case)))
        (report-on-failure case))))
```

It is a (unproved!!) Common Lisp (not ACL2) program!!

A concrete proposal

To obtain the Eilenberg-Zilber theorem automated proof using ACL2

Challenge:

- it is an important theorem implemented in a Kenzo modul used by the system

- its feasibility is not secure

An example

Formalizing Simplicial Topology in ACL2

Mirian Andrés
Laureano Lambán
Julio Rubio

José Luis Ruiz Reina

University of La Rioja (U.R.), Spain

University of Seville (U.S.), Spain

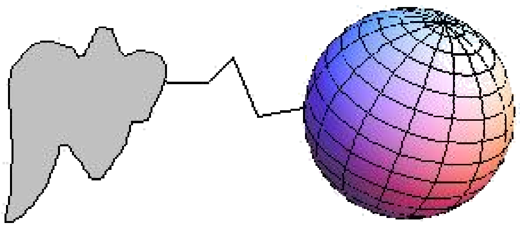
ACL2 WORKSHOP 2007, Austin (Texas)
November 15th-16th, 2007

Simplicial Topology in ACL2

Abstract topological spaces replaced by simplicial sets (combinatorial artifacts)

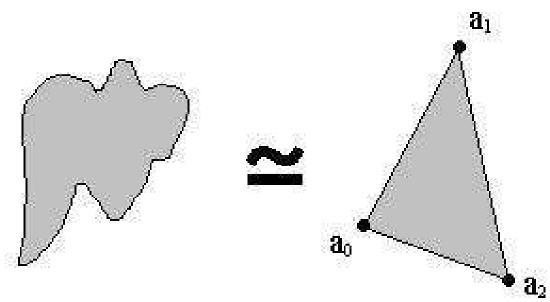
- Motivation: algebraic invariants are computed in an easier way

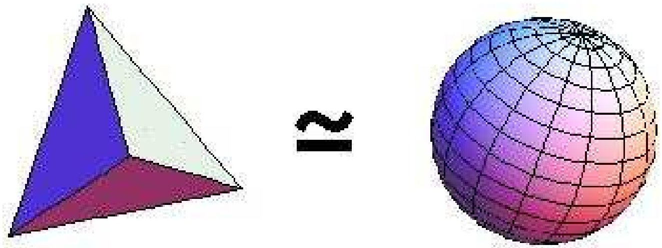
Example: topological space



Simplicial Topology in ACL2

Triangulating the space





Triangle can be described by (a_0, a_1, a_2) where the faces are obtained in this way:

$$\partial_0(a_0,a_1,a_2)=(a_1,a_2)$$
$$\partial_1(a_0,a_1,a_2)=(a_0,a_2)$$
$$\partial_2(a_0,a_1,a_2)=(a_0,a_1)$$

$$\partial_i \, \partial_j = \partial_{j-1} \, \partial_i \quad \textbf{if } i<j$$

The faces of each edge are defined analogously :

$$\partial_0(a_1,a_2)=(a_2)$$
$$\partial_1(a_1,a_2)=(a_1)$$

.

.

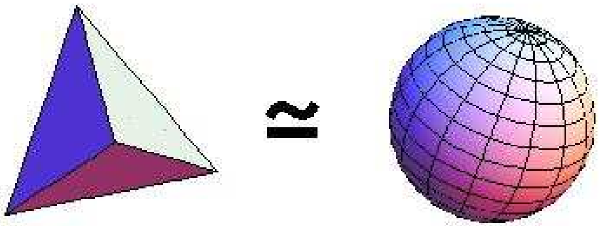
.

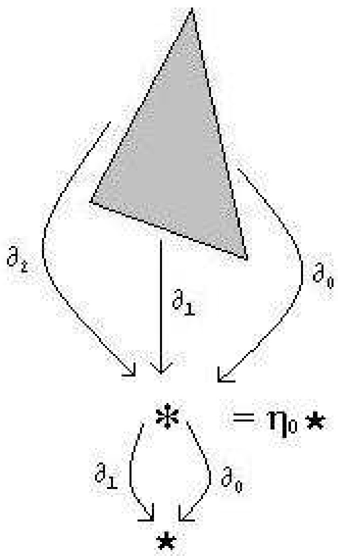
Simplicial Topology in ACL2

4 vertices

6 edges } **14 elements**

4 triangles





Triangle faces:

$\partial_0 x = \partial_1 x = \partial_2 x = \eta_0(*)$.

1 triangle

1 collapsing point } **2 elements**

$\eta_0(*)$ is called **degeneration** of $\star$

Simplicial Topology in ACL2

$$\eta_0 (a_0, a_{1,}, a_2) := (a_0 , a_0 , a_1 , a_2)$$

$$\eta_1 (a_0, a_{1,}, a_2) := (a_0 , a_1 , a_1 , a_2)$$

$$\eta_2 (a_0, a_{1,}, a_2) := (a_0 , a_1 , a_2 , a_2)$$

The operator η_i is repeating the i-th element in the list

Simplicial Topology in ACL2

Definition. A ***simplicial set*** K consists of a graded set $\{K_q\}_{q \in N}$ and, for each pair of integers (i,q) with 0<=i<=q, *face* and *degeneracy* maps, $\partial_i : K_q \rightarrow K_{q-1}$ and $\eta_i : K_q \rightarrow K_{q+1}$, satisfying the simplicial identities:

$$\partial_i \partial_j = \partial_{j-1} \partial_i \quad \text{if } i<j$$
$$\eta_i \eta_j = \eta_{j+1} \eta_i \quad \text{if } i<=j$$
$$\partial_i \eta_j = \eta_{j-1} \partial_i \quad \text{if } i<j$$
$$\partial_i \eta_j = Id \quad \text{if } i=j \text{ or } i=j+1$$
$$\partial_i \eta_j = \eta_j \partial_{i-1} \quad \text{if } i>j+1$$

The elements of K_q are called ***q-simplices***

A q-simplex x is **degenerate** if $x = \eta_i y$ with $y \in K_{q-1}$, 0<=i<q

Otherwise x is called **non-degenerate**

0-simplices as vertices
Non-degenerate 1-simplices as edges
Non-degenerate 2-simplices as (filled) triangles
Non-degenerate 3-simplices as (filled) tetrahedra
...

Simplicial Topology in ACL2

We focus our studies on the **universal simplicial set Δ**

- *Reason*: Any theorem proved on Δ by using only the equalities of the previous definition will be also true for any other simplicial set K

In ACL2

- a q-simplex of Δ is any ACL2 list of length q
- face operators are defined by means of the function `(del-nth i l)` which eliminates the i-th element in the list l
- degeneracy operators are defined by means of the function `(deg i l)` which repeats the i-th element in the list l

We consider the simplicial set freely generated from the set of all ACL2 objects

A developed example

Theorem 1. Let K be a simplicial set. Any degenerate n-simplex x ∈ K_n can be expressed in a unique way as a (possibly) iterated degeneracy of a non-degenerate simplex y in the following way:

$$x = \eta_{j_k} \dots \eta_{j_1} y$$

with y ∈ K_r , k = n-r > 0, $0 <= j_1 < ... <= j_k < n$

Thinking in ACL2

- A non-degenerate simplex in Δ is a list where any two consecutive elements are different
- A simplex in Δ can be represented as a pair of lists, the first one a list of natural numbers (degeneracy list) and the second one any ACL2 list.

Theorem 2. Any ACL2 list l can be expressed in a unique way as a pair (dl,l') such that l= degenerate (dl,l') with l' without two consecutive elements equal and dl a strictly increasing degeneracy list.

A direct ACL2 proof of theorem 2

```
(defun generate (l)
  (if (or (endp l) (endp (cdr l)))
            (cons nil l)
    (let ((gencdr (generate (cdr l))))
      (if (equal (first l) (second l))
          (cons (cons 0 (add-one (car gencdr)))
                (cdr gencdr))
        (cons (add-one (car gencdr))
              (cons (car l) (cdr gencdr)))))))

(defthm existence
  (let ((gen (generate l)))
    (and (canonical gen)
         (equal (degenerate (car gen) (cdr gen)) l))))
```

A direct ACL2 proof of theorem 2

```
(defthm uniqueness-main-lemma
  (implies (canonical (cons l1 l2))
           (equal (generate (degenerate l1 l2))
                  (cons l1 l2))))
```

The lists obtained after rewriting `(generate (degenerate l1 l2))` in `(generate (degenerate (cdr l1) (deg (car l1) l2)))` do not satisfy the hypotheses of the theorem. Not possible to apply a simplified induction scheme.

```
(defthm uniqueness
  (implies
    (and (canonical p1) (canonical p2)
         (equal (degenerate (car p1) (cdr p1)) l)
         (equal (degenerate (car p2) (cdr p2)) l))
    (equal p1 p2)))
```

An abstract reduction systems approach

An alternative proof because:

- The direct proof does not explicitly use the face operators
- The direct proof is not directly based on the combinatorial properties which relate the face and degeneracy maps

Idea:

To consider the elimination of a consecutive repetition in a list (face operator) as a simple **reduction step**

Another type of **reduction step** to “fix” disorders in the degeneracy list

An abstract reduction systems approach

Formalizing:

- We define the reduction system $\rightarrow_S$ where:
  - the set of *S*-terms is the set of pairs (l_1, l_2) where

    l_1 a list of natural numbers

    l_2 any list
  - two types of rules are considered in $\rightarrow_S$:

• ***o-reduction***: if the list l_1 has a "disorder" at position i, i.e., $l_1(i) >= l_1(i+1)$, then $(l_1, l_2) \rightarrow_S (l'_1, l_2)$, where $l'_1(i) = l_1(i+1)$ and $l'_1(i+1) = l_1(i)+1$, (here l(j) denotes the j-th element of l)

$$\eta_i \, \eta_j = \eta_{j+1} \, \eta_i \qquad \texttt{if } i<=j$$

•***r-reduction***: if at index i there is a repetition in l_2 , i.e. , $l_2(i) = l_2(i+1)$, then $(l_1, l_2) \rightarrow_S (l'_1, l'_2)$, where l'_1=cons(i, l_1) and l'_2 =del-nth (i, l_2)

$$\partial_i \, \eta_j = \mathrm{Id} \qquad \texttt{if } i=j \texttt{ or } i=j+1$$

An abstract reduction systems approach

- Modeling our reduction system in ACL2

- Model $\rightarrow_S$ in the framework of **Ruiz Reina's** ACL2 formalization about abstract reduction systems

Operators are pairs (t,i) where
t is ‘o or ‘r
i is the position in the list where the corresponding reduction takes place

The **relation** $\rightarrow_S$ is represented by two functions :

```
(s-legal x op)
(s-reduce-one-step x op)
```

They suffice to represent a reduction and other related concepts:
noetherianity, equivalence closures, normal forms or confluence

An abstract reduction systems approach

- We proved that the reduction is noetherian *(there is no infinite sequence of S-reductions)* using a suitable lexicographic measure

- We defined a function to compute a normal form with respect to $\rightarrow_S$

```
(defun s-normal-form (x)
  (let ((red (s-reducible x)))
    (if red
          (s-normal-form (s-reduce-one-step x red))
      x)))
```

- We proved that $\rightarrow_S$ is locally confluent *(whenever there is a local peak, there is a valley)*

```
(defthm local-confluence
  (implies (and (s-equiv-p x y p) (local-peak-p p))
     (and (s-equiv-p x y (s-transform-local-peak p))
         (steps-valley (s-transform-local-peak p)))))
```

- Newman's Lemma: every noetherian and locally confluent reduction is convergent. It means that two equivalent elements have a common normal form

```
(defthm s-reduction-convergent
  (implies (s-equiv-p x y p)
     (equal (s-normal-form x) (s-normal-form y))))
```

An abstract reduction systems approach

- The main relation between $\rightarrow_S$ and the function degenerate is given by

  a) If $(l_1, l_2) \rightarrow_S (l_3, l_4)$, then degenerate (l_1, l_2)= degenerate (l_3, l_4)

  b) If degenerate (l_1, l_2)=l then (nil,l)$=_S (l_1, l_2)$

```
(defthm degenerate-s-equivalent
   (implies …
      (s-equiv-p (cons l m)
                         (cons nil (degenerate l m))
                         (degenerate-steps l m))))
```

- We define (generate l) as (s-normal-form (cons nil l)))

- We prove the theorems existence and uniqueness exactly as stated previously

- Corollary: both definitions of generate are equivalent

Conclusions

- We have presented some ideas to apply ACL2 in Simplicial Topology. Main contributions:
  - ✓analysis of feasibility
  - ✓relation of ACL2 proofs in Simplicial Topology with abstract rewriting systems
- Increase the reliability of a real Computer Algebra program (Kenzo)

Further work

- Formalize and prove more difficult results from Simplicial Topology in ACL2
- ACL2 proof of the Eilenberg-Zilber theorem

Automated theorem proving in Simplicial Topology with ACL2

Mirian Andrés

University of La Rioja (U.R.), Spain

ICTP, Trieste (Italy)
2008 August