HOMENAJE A MIRIAN

- Introducción
- Imágenes para (intentar) no estar tristes
- Demo de ACL2
- Verificación formal y topología algebraica

EACA 2004 (SANTANDER)

EACA 2004 (SANTANDER)

EACA 2004 (SANTANDER)

EACA 2004 (SANTANDER)

EACA 2004 (SANTANDER)

SEVILLA 2005

SEVILLA 2005

SEVILLA 2005

SEVILLA 2005

ACL2 2007 (AUSTIN, TX)

ACL2 2007 (AUSTIN,TX)

NUEVA YORK 2007

Demostración automática

- Un demostrador automático es un programa que puede encontrar/comprobar demostraciones en un sistema lógico
  - Sus demostraciones alcanzan tal nivel de detalle que se suelen denominar “pruebas formales”, en contraste con las demostraciones “a mano”.
- ¿Por qué es interesante formalizar?
  - Descripciones formales y rigurosas de los sistemas (hardware, software, matemáticas,...)
  - Demostraciones formales de sus propiedades
- Por tanto, se incrementa la confianza en los sistemas
  - Un sistema formalizado *es más fiable*

ACL2

- A Computational Logic for an Applicative Common Lisp
- ACL2 es:
  - Un lenguaje de programación (una extensión de un subconjunto aplicativo de Common Lisp)
  - Una lógica (de primer orden), que permite especificar y demostrar propiedades acerca de los programas
  - Un demostrador automático que asiste al usuario en la búsqueda y comprobación de esas demostraciones

DEMOSTRANDO CON ACL2

- ACL2 es automático:
  - Una vez se inicia un intento de demostración, no hay interacción del usuario
  - Técnicas: inducción y reescritura
- Pero interactivo en un sentido más profundo:
  - El usuario “guía” al demostrador hacia una prueba preconcebida, mediante un conjunto de lemas previos que se usan como reglas de reescritura
- El papel del usuario:
  - Expresar las propiedades en la lógica
  - Guiar hacia una prueba preconcebida

ACL2 DEMO....

VERIFICACIÓN DE HARDWARE

- **Modelo formal del microcódigo de la división del procesador AMD K5 (testeado):**

```
(defun divide (p d mode)
  ....  ;;; cientos de líneas de código ACL2
  ....  ;;; describiendo fielmente el microcódigo
  ....)
```

- **Verificación formal:**

```
(defun AMD-K5-division-correct
   (implies ....
            (equal (divide p d mode)
                   (round (/ p d) mode))))
```

¿Podemos hacer algo análogo con los sistemas de álgebra computacional?

Formalizando la topología simplicial en ACL2

Mirian Andrés
Laureano Lambán
Julio Rubio

Universidad de La Rioja (U.R.)

José Luis Ruiz Reina

Universidad de Sevilla (U.S.)

Universidad de Sevilla
8 de Julio de 2008

ÍNDICE

- Introducción (Kenzo)
- Nuestro objetivo general
- Un objetivo concreto
- Un ejemplo

  Topología Simplicial en ACL2

  Un ejemplo desarrollado

  Demostración directa

  Demostración basada en sistemas de reducción abstractos

- Conclusiones y trabajo futuro

Introducción

El sistema en Common Lisp

Kenzo

Para hacer cálculos en Topología Algebraica

- testeado pero … no siempre

¡¡¡ No se ha demostrado la corrección de sus programas !!!

Ahora nos concentramos en intentar incrementar su fiabilidad

Objetivo general

- demostradores de teoremas para hacer testeo automático de programas

- primero demostrar y después ... testear

- demostraciones mecanizadas en Isabelle para algunos algoritmos teóricos usados en **Kenzo**

(Demostración en Isabelle del Lema de Perturbación Básico desarrollada por J. M. Aransay)

- hay una distancia entre el código de Kenzo y las teorías y demostraciones en Isabelle

Objetivo general

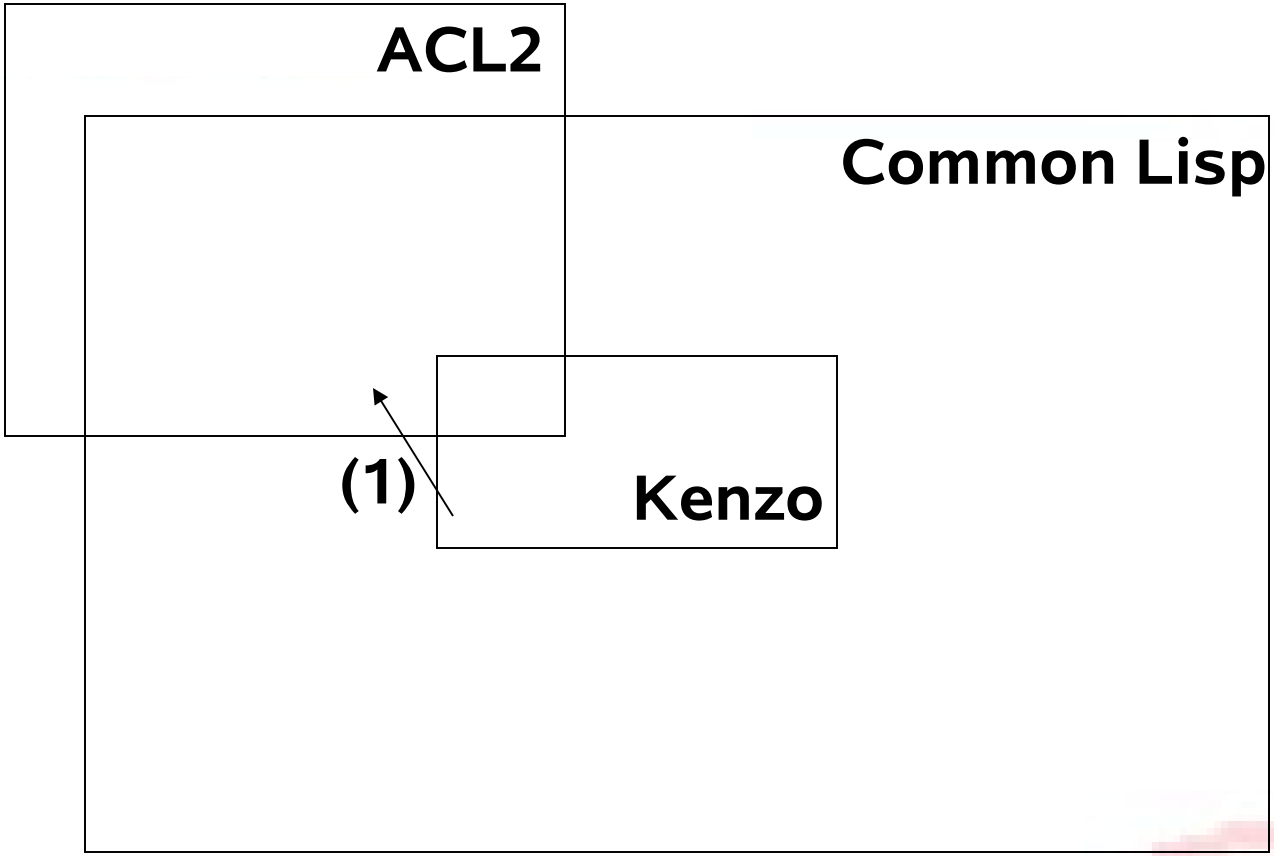
Asumimos un programa Common Lisp
`programa1`
con las siguientes características:

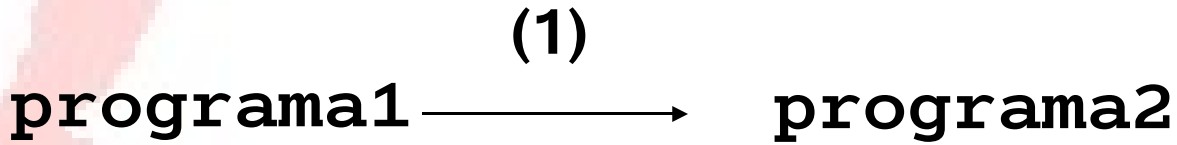
- difícil de testear
(por resultados difíciles de obtener o desconocidos)

- difícil de demostrar
(por ejemplo, es lógicamente complejo:
de orden superior)

ACL2: A Computational Logic for Applicative Common Lisp

ACL2 es una extensión de una parte de Common Lisp





programa1 es

- ya está escrito
- Common Lisp (no ACL2)
- eficiente
- testeado
- no demostrado

programa2 es

- diseñado especialmente para ser demostrado
- ACL2 (y Common Lisp)
- eficiente o no : irrelevante
- testeado
- demostrado en ACL2

programa2

"se ***supone*** equivalente a"

programa1

no esperamos ***demostrar*** la equivalencia

Pero la usamos para hacer ***testeo automático***

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

Es un programa (no demostrado) en Common Lisp
pero no es un programa ACL2

Objetivo general

Nuestra idea: usar ACL2 para verificar los programas en Kenzo

Pero ... Kenzo usa programación funcional de orden superior

¿Cómo podemos hacer Kenzo más fiable?

Nuestro propósito:

Elegir, reprogramar y verificar en ACL2 fragmentos de Kenzo de primer orden, relacionados con Topología Simplicial

Un objetivo concreto

Conseguir la demostración automática del teorema de

Eilenberg-Zilber utilizando ACL2

Reto:

- es un teorema importante implementado en un módulo de Kenzo y usado por el sistema

- se duda de la viabilidad de la obtención de este objetivo

- se intenta conseguir con este trabajo una tesis doctoral

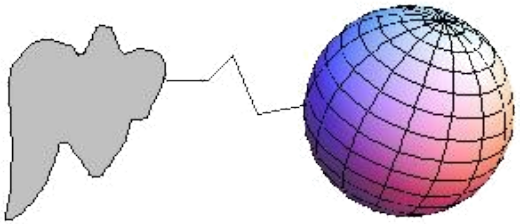
Un ejemplo

Topología Simplicial en ACL2

Espacios Topológicos Abstractos reemplazados por Conjuntos Simpliciales (artefactos combinatorios)

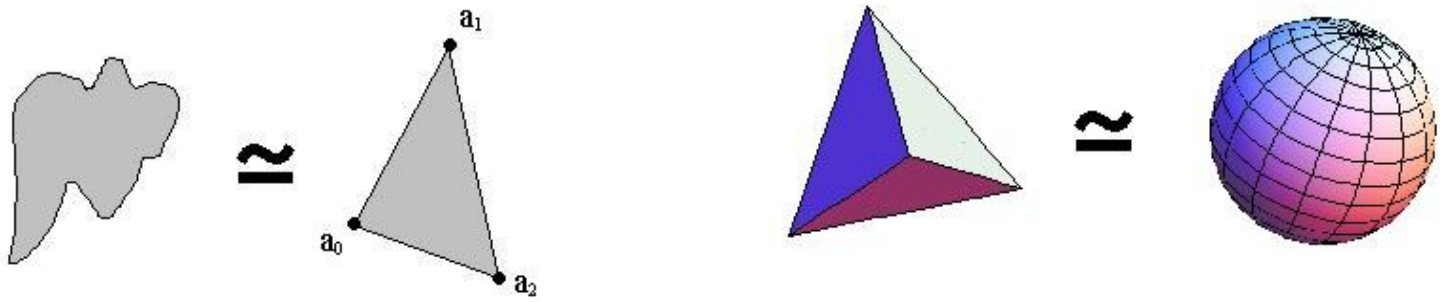
- Motivación: invariantes algebraicos son computados de una forma más sencilla

Ejemplo: espacio topológico



Topología Simplicial en ACL2

Triangulando el espacio



Triángulo se puede describir (a_0, a_1, a_2) donde las caras se obtienen del siguiente modo:

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

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

Las caras de cada segmento se definen de forma análoga :

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

.

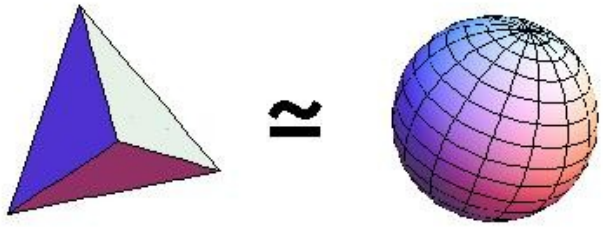
.

.

Topología Simplicial en ACL2

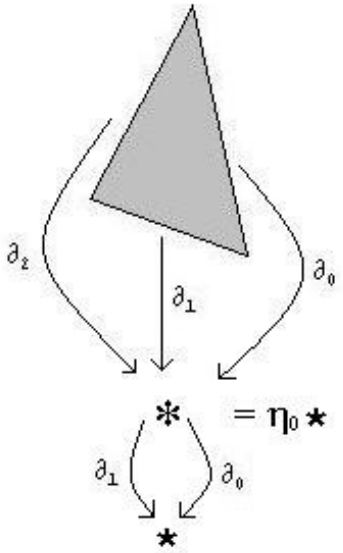
4 vértices
6 segmentos
4 triángulos

} 14 elementos



Caras del triángulo:

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



1 triángulo

1 punto de colapso

} 2 elementos

$\eta_0(*)$ la llamamos **degeneración** de □

Topología Simplicial en ACL2

$$\eta_0(a_0, a_1, a_2) := (a_0, a_0, a_1, a_2)$$

$$\eta_1(a_0, a_1, a_2) := (a_0, a_1, a_1, a_2)$$

$$\eta_2(a_0, a_1, a_2) := (a_0, a_1, a_2, a_2)$$

El operador η_i repite el elemento i-ésimo en la lista

Topología Simplicial en ACL2

Definición. Un ***conjunto simplicial*** K consiste en un conjunto graduado $\{K_q\}_{q \in N}$ y para cada par de enteros (i,q) con 0<=i<=q, aplicaciones llamadas *cara* y *degeneración*, $\partial_i : K_q \to K_{q-1}$ y $\eta_i : K_q \to K_{q+1}$, que satisfacen las identidades simpliciales siguientes:

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

Los elementos de K_q se llaman ***q-símplices***

Un q-simplex x es **degenerado** si $x = \eta_i y$ con $y \in K_{q-1}$, $0<=i<q$

En otro caso x se dice **no-degenerado**

0-simplices no degenerados como vértices
1-simplices no degenerados como segmentos
2-simplices no degenarados como triángulos (rellenos)
3-simplices no degenerados como tetraedros (rellenos)
...

Topología Simplicial en ACL2

Centramos nuestros estudios en el **conjunto universal simplicial Δ**

- *Razón*: cualquier teorema demostrado en Δ usando sólo las igualdades

de la definición previa es cierta también para cualquier otro conjunto simplicial k

En ACL2

- un q-símplice de Δ es cualquier lista ACL2 de longitud q
- los operadores cara se definen con la función `(del-nth i l)` que elimina el elemento i-ésimo de la lista l
- los operadores degeneración se definen con la función `(deg i l)` que repite el elemento i-ésimo en la lista l

Consideramos el conjunto simplicial libremente generado del conjunto de todos los objetos ACL2

Un ejemplo

Teorema 1. Sea K un conjunto simplicial. Cualquier n-símplice degenerado $x \in K_n$ se puede expresar de forma única como (posiblemente) degeneraciones iteradas de un símplice y no degenerado en el siguiente modo:

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

con $y \in K_r$, $k = n-r > 0$, $0 <= j_1 < ... <= j_k < n$

Pensando en ACL2

- Un símplice no degenerado en Δ es una lista donde cualesquiera dos elementos consecutivos son diferentes
- Un símplice en Δ se puede representar como un par de listas, la primera una lista de números naturales (lista de degeneración) y la segunda, cualquier lista ACL2

Teorema 2. Cualquier lista l ACL2 se puede expresar de forma única como un par (dl,l') tal que l= degenerate (dl,l') con l' sin dos elementos consecutivos iguales y dl una lista degenerada estrictamente creciente.

Una demostración directa en ACL2 del teorema 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))))
```

Una demostración directa en ACL2 del teorema 2

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

Las listas obtenidas después de reescribir `(generate (degenerate l1 l2))` en `(generate (degenerate (cdr l1) (deg (car l1) l2)))` no satisfacen las hipótesis del teorema. No es posible aplicar un esquema simplificado de inducción.

```
(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)))
```

Una aproximación con sistemas de reducción

Una prueba alternativa porque:

- La prueba directa no usa explícitamente los operadores cara
- La prueba directa no se basa directamente en las propiedades combinatorias que relacionan las aplicaciones cara y degeneración

Idea:

Considerar la eliminación de una repetición consecutiva en una lista (operador cara) como un **paso simple de reducción**

Otro tipo de **paso de reducción** para eliminar desórdenes en la lista de degeneración

Una aproximación con sistemas de reducción

Formalizando:

- Definimos el sistema de reducción $\rightarrow_S$ donde:
  - el conjunto de *S*-términos es el conjunto de pares (l_1, l_2) donde

    l_1 es una lista de números naturales

    l_2 es cualquier lista
  - dos tipos de reglas se consideran en $\rightarrow_S$:

- ***o-reducción***: si la lista l_1 tiene un "desorden" en la posición i, es decir, $l_1(i) >= l_1(i+1)$, entonces $(l_1, l_2) \rightarrow_S (l'_1, l_2)$, donde $l'_1(i) = l_1(i+1)$ y $l'_1(i+1) = l_1(i)+1$, (aquí l(j) denota el elemento j-ésimo de l)

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

- ***r-reducción***: si hay una repetición en el índice i de l_2 ,es decir , $l_2(i) = l_2(i+1)$, entonces $(l_1, l_2) \rightarrow_S (l'_1, l'_2)$, donde l'_1=cons(i, l_1) y l'_2 =del-nth (i, l_2)

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

Una aproximación con sistemas de reducción

- Modelando nuestro sistema de reducción en ACL2

- Modelar $\rightarrow_S$ en el marco de la formalización ACL2 de **J.L. Ruiz Reina** sobre sistemas de reducció

Operadores son pares (t,i) donde
t es ‘o or ‘r
i es la posición de la lista donde tiene lugar la reducción correspondiente

La **relation** $\rightarrow_S$ se representa con dos funciones :

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

Estas operaciones son suficientes para representar una reducción y otros conceptos relacionados:
noetherianidad, clausuras de equivalencia, formas normales o confluencia

Una aproximación con sistemas de reducción

- Demostramos que la reducción es noetheriana *(no hay infinitas secuencias de S-reducciones)*
usando una medida lexicográfica apropiada

- Definimos una función para calcualr una forma normal respecto a $\rightarrow_S$

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

- Demostramos que $\rightarrow_S$ es localmente confluente *(donde hay un pico local se puede construir un valle)*

```
(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)))))
```

- Lema de Newman: toda reducción noeteriana y localmente confluente es convergente.
Significa que dos elementos equivalentes tienen una forma normal común

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

Una aproximación con sistemas de reducción

- La relación principal entre $\rightarrow_S$ y la función degeneración son dadas por:
  - Si $(l_1, l_2) \rightarrow_S (l_3, l_4)$,entonces degenerate (l_1, l_2)= degenerate (l_3, l_4)
  - Si degenerate (l_1, l_2)=l entonces (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))))
```

- Definimos (generate l) como (s-normal-form (cons nil l)))

- Demostramos los teoremas de existencia y unicidad exactamente como los hemos establecido previamente

- Corolario: ambas definiciones de generar son equivalentes

Conclusiones

- Hemos presentados algunas ideas para aplicar ACL2 en Topología Simplicial.

Contribuciones principales:

  - ✓ analisis de viabilidad
  - ✓ relación de demostraciones ACL2 en TopologíaSimplicial sistemas abstractos de reduccion

- Incrementar la fiabilidad de un sistema real de álgebra computacional (Kenzo)

Trabajo futuro

- Formalizar y demostrar resultados más complicados de Topología Simplicial en ACL2
- Demostración en ACL2 del teorema de Eilenberg-Zilber

El papel de ACL2 para incrementar la fiabilidad de Kenzo

Mirian Andrés
Laureano Lambán
Julio Rubio

Universidad de La Rioja (U.R.)

José Luis Ruiz Reina

Universidad de Sevilla (U.S.)

Universidad de Sevilla
8 de Julio de 2008