ACL2: Introducción

Francisco J. Martín Mateos
José L. Ruiz Reina

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

¿Qué es ACL2?

ACL2 es un demostrador interactivo de teoremas para una lógica computacional de primer orden sin cuantificación explícita y con igualdad

- El lenguaje es una extensión de un subconjunto aplicativo de Common Lisp
- La demostración se hace de forma directa: se intenta comprobar un resultado a partir de definiciones, axiomas y lemas previos
- El razonamiento se realiza principalmente por inducción estructural y simplificación por reescritura. También usa otras estrategias de prueba como simplificación por congruencias, encadenamiento hacia adelante, generalización, fertilización cruzada, instanciación e instanciación funcional
- Las pruebas son una descripción detallada en lenguaje natural (inglés) de cómo se llega a demostrar un resultado utilizando las estrategias de prueba del sistema

¿Qué es ACL2?

- ACL2 es
  - Un lenguaje de programación
  - Una lógica que permite razonar sobre programas
  - Un demostrador automático que ayuda en la demostración de propiedades en la lógica
- Con ACL2 se puede
  - Construir modelos formales en el lenguaje de programación
  - Evaluar dichos modelos y comprobar su funcionamiento
  - Demostrar propiedades sobre los modelos

¿Qué es ACL2?

- Desarrollado por Matt Kaufmann y J S. Moore en el Departamento de Ciencias de la Computación de la Universidad de Texas en Austin
- Aplicaciones
  - Modelado de procesadores y verificación de hardware
  - Lenguajes de programación y verificación de software
  - Aritmética de punto flotante y aritmética real
  - Problemas de concurrencia

Algunos trabajos relevantes en ACL2

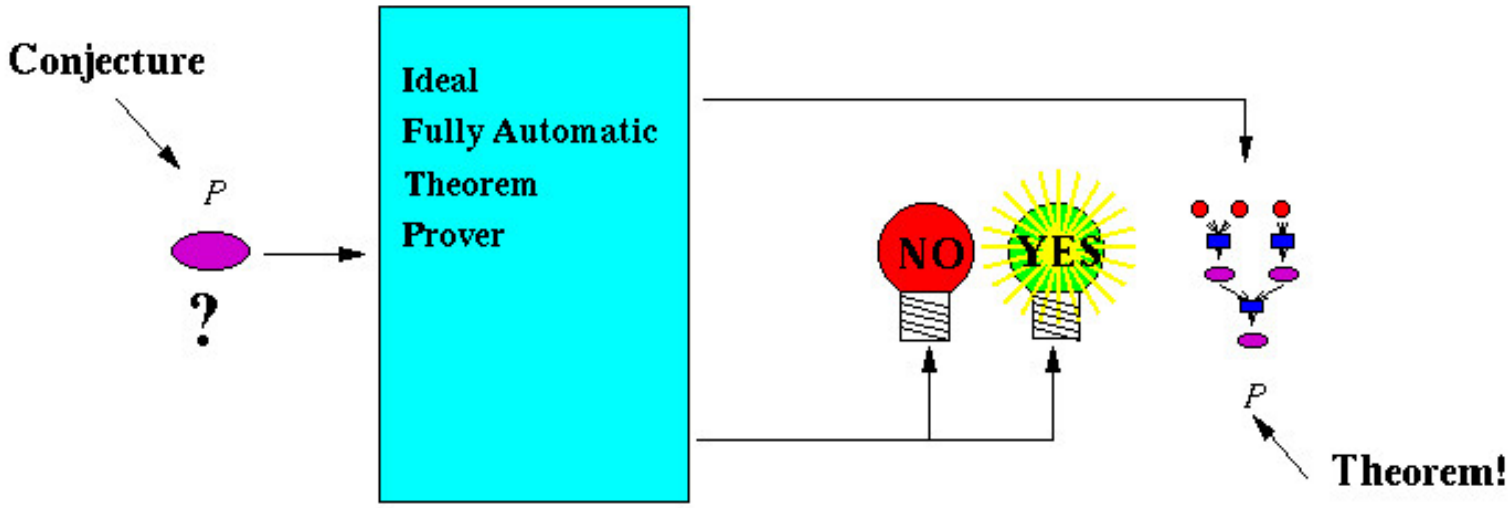
- Matemáticas y lógica
  - Sistemas de reescritura
  - Lemas de Dickson y Higman
  - Algoritmo de Buchberger
  - Sistemas de demostración automática proposicionales
  - Verificación de pruebas de OTTER
  - Primer teorema de incompletitud de Gödel

Algunos trabajos relevantes en ACL2

- Verificación de *hardware* y *software*
  - La "pila CLI", un sistema académico que incluye microprocesador, enlazador, ensamblador, compilador y aplicaciones
  - Verificación de bytecode Java
  - 21 de las 22 rutinas de la biblioteca de cadenas de Berkeley C (compiladas para el procesador Motorola 6820)
  - Programas de microcódigo de la ROM del procesador de señales digitales CAP, de Motorola
  - El microcódigo del algoritmo de división de coma flotante y de raíz cuadrada del procesador AMD K5
  - El código RTL que implementa las operaciones elementales sobre números de coma flotante del procesador AMD Athlon

ACL2 como demostrador interactivo de teoremas

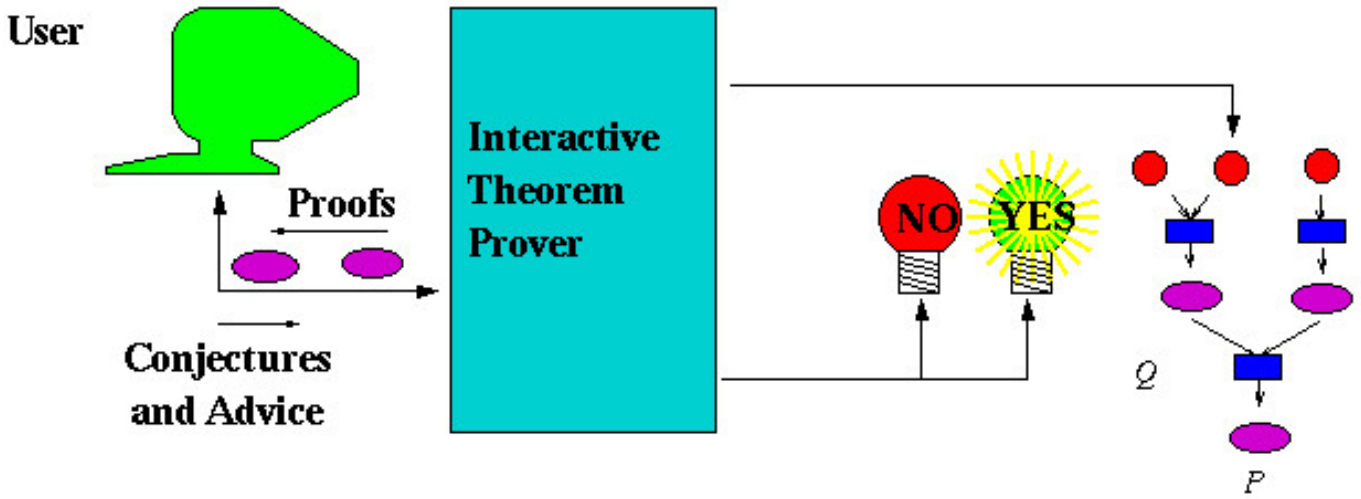
- Un demostrador automático de teoremas ideal proporciona una prueba de un resultado, asegurando que es cierto, o confirma que no existe dicha prueba, indicando que el resultado es falso



- Este tipo de demostradores sólo se pueden construir para lógicas muy simples

ACL2 como demostrador interactivo de teoremas

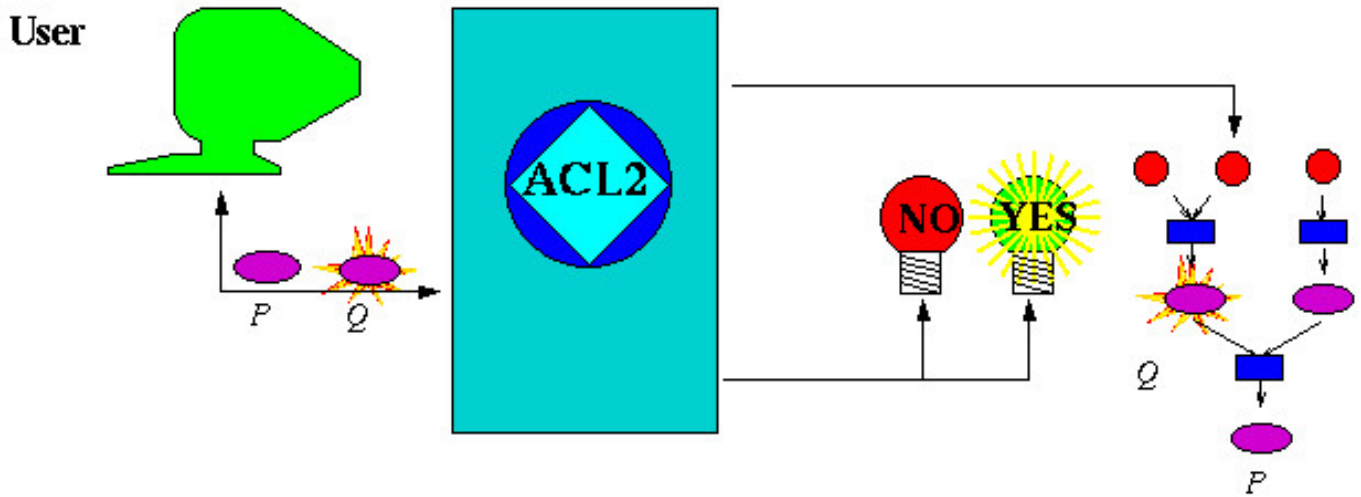
- Un demostrador interactivo de teoremas requiere del usuario para la obtención de la prueba de un resultado



- El usuario interactúa en el proceso de la prueba indicando los pasos a seguir

ACL2 como demostrador interactivo de teoremas

- ACL2 es automático en el sentido de que una vez comenzado un intento de prueba, el usuario no puede interactuar con el sistema, salvo para cancelar el proceso
- ACL2 es interactivo en el sentido de que el usuario puede guiar al sistema demostrando resultados previos que permitan encontrar una prueba de forma automática



Instalación y ejecución

- Descargar el código fuente de la página del sistema
  - Precompilado
    - Versión 8.0:
      http://acl2s.ccs.neu.edu/acl2s/src/acl2/
    - Versión 3.6 para Windows:
      https://www.cs.utexas.edu/users/moore/acl2/v3-6/distrib/windows/
  - Acl2 Sedan: Entorno de ejecución integrado en eclipse

Instalación y ejecución

- Ejecución integrada en Emacs como un intérprete de Lisp
  - Ficheros con extensión `.lisp`
- Fichero de configuración de Emacs
  - En Windows: `C:\Users\username\AppData\Roaming\.emacs`
  - En Windows con Emacs: `~/.emacs`
  - En Linux: `.emacs`
- Añadir al fichero de configuración de Emacs

```
;;;-------------------------------------------------

(defun run-acl2 ()
  (interactive)
  (run-lisp "/camino/completo/ejecutable/acl2"))

;;;-------------------------------------------------
```

El lenguaje de programación

- Una *extensión* de un *subconjunto aplicativo* de Common Lisp
  - Esto permite compilación y ejecución en *cualquier* Common Lisp
- Características propias de Common Lisp
  - Notación prefija; función y argumentos

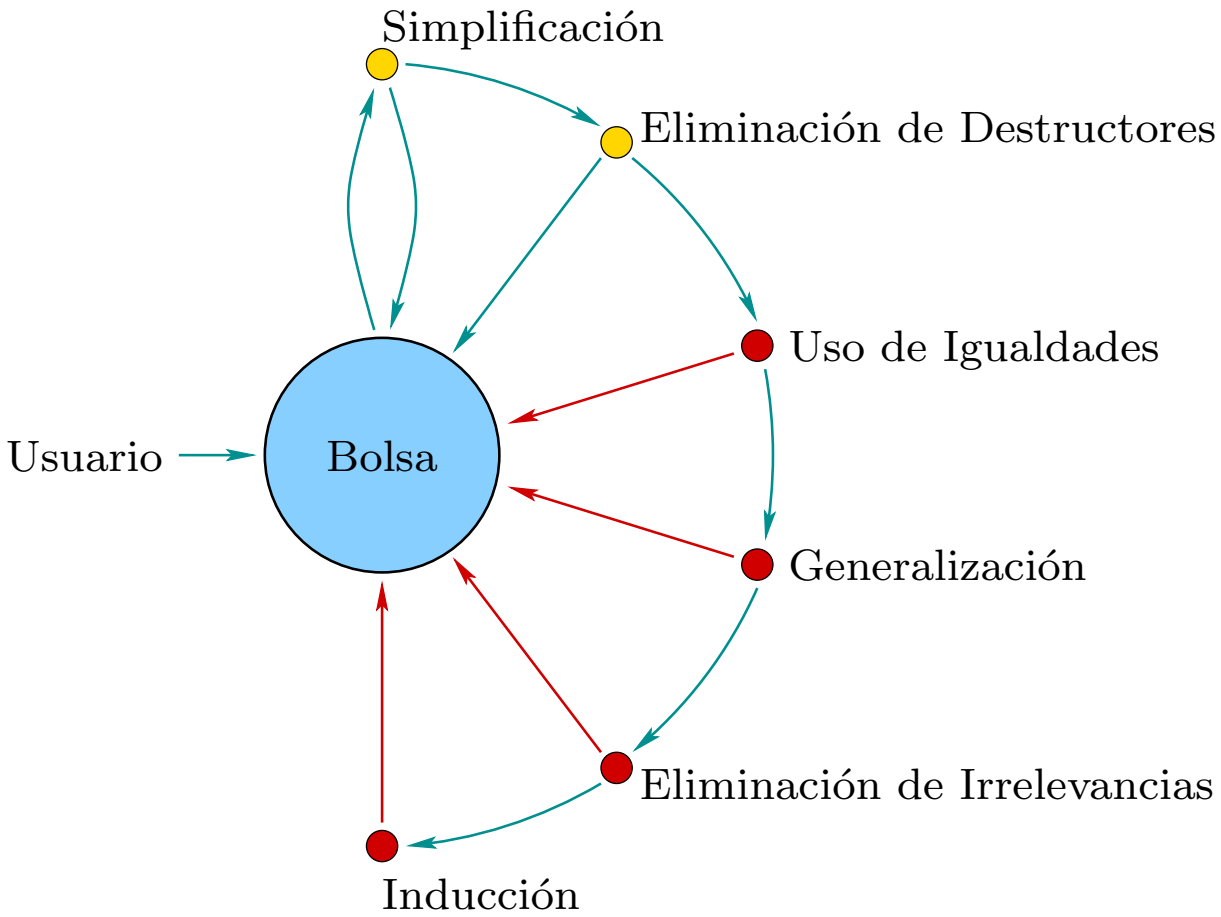
    `(f a1 ... an)`
  - Expresiones anidadas, paréntesis, listas y átomos

    `'(a (b) c)`
  - Lenguaje interpretado: bucle lee, evalúa, imprime

El lenguaje de programación

- Pares ordenados
  - **`cons`**, **`car`**, **`cdr`**, **`consp`**
- Listas
  - **`nil`**, **`endp`**, **`list`**, **`len`**, **`member`**, **`last`**, **`append`**, **`reverse`**
- Lógica
  - **`nil`**, **`t`**, **`not`**, **`and`**, **`or`**, **`equal`**
- Aritmética
  - **`+`**, **`-`**, **`*`**, **`/`**, **`<`**, **`<=`**, **`=`**, **`>`**, **`>=`**
- Condicionales
  - **`if`**, **`cond`**
- Definiciones
  - **`defun`**

organización del demostrador



Organización del demostrador

- Funcionamiento del demostrador
  - La conjetura inicial proporcionada por el usuario se introduce en la "bolsa"
  - El sistema trata de aplicar distintos procesos a las conjeturas de la "bolsa"
  - Si un proceso es aplicable a una conjetura, el resultado se introduce de nuevo en la "bolsa"
  - Si un proceso no es aplicable a una conjetura, entonces se pasa al siguiente proceso
- Posibles situaciones finales
  - La "bolsa" queda vacía: el objetivo inicial es teorema
  - El sistema para y la "bolsa" no está vacía: resultado desconocido
  - Es posible entrar en ciclos !!

Organización del demostrador

- Características de los procesos aplicados
  - Reciben una fórmula (objetivo) y devuelven un conjunto de fórmulas (subobjetivos)
  - Si los subobjetivos son ciertos entonces el objetivo también
  - Sólo los dos primeros preservan la equivalencia
  - El resto son procesos "peligrosos"
  - La aplicación de estos procesos no es reversible, si un proceso es aplicado con éxito entonces no hay vuelta atrás
  - Si un proceso es aplicado con éxito entonces la explicación del mismo es impresa en pantalla

Ejemplo

- Implementar un procedimiento para invertir una lista y demostrar que la inversa de la concatenación de dos listas $\mathbf{l_1}$ y $\mathbf{l_2}$ es igual a la concatenación de la inversa de $\mathbf{l_2}$ con la inversa de $\mathbf{l_1}$

```
(defun concatena (l1 l2)
  (if (endp l1)
      l2
      (cons (car l1) (concatena (cdr l1) l2))))

(defun inversa (l)
  (if (endp l)
      nil
      (concatena (inversa (cdr l)) (list (car l)))))
```

Ejemplo

- Análisis de admisibilidad de la función **CONCATENA**

```
The admission of CONCATENA is trivial, using the relation O< (which
is known to be well-founded on the domain recognized by O-P) and the
measure (ACL2-COUNT L1).  We observe that the type of CONCATENA is
described by the theorem
(OR (CONSP (CONCATENA L1 L2)) (EQUAL (CONCATENA L1 L2) L2)).  We used
primitive type reasoning.

Summary
Form:  ( DEFUN CONCATENA ...)
Rules: ((:FAKE-RUNE-FOR-TYPE-SET NIL))
Time:  0.01 seconds (prove: 0.00, print: 0.00, other: 0.00)
 CONCATENA
```

Ejemplo

- Análisis de admisibilidad de la función **INVERSA**

```
The admission of INVERSA is trivial, using the relation O< (which is
known to be well-founded on the domain recognized by O-P) and the measure
(ACL2-COUNT L).  We observe that the type of INVERSA is described by
the theorem (OR (CONSP (INVERSA L)) (EQUAL (INVERSA L) NIL)).  We used
primitive type reasoning and the :type-prescription rule CONCATENA.

Summary
Form:  ( DEFUN INVERSA ...)
Rules: ((:FAKE-RUNE-FOR-TYPE-SET NIL)
        (:TYPE-PRESCRIPTION CONCATENA))
Time:  0.00 seconds (prove: 0.00, print: 0.00, other: 0.00)
 INVERSA
```

Ejemplo

- Ejecución

```
ACL2 !>(inversa '(a 1 b 4))
(4 B 1 A)
ACL2 !>(inversa '(1 2 3 4 5 6 7 8 9 10))
(10 9 8 7 6 5 4 3 2 1)
ACL2 !>(inversa (inversa '(a 1 b 4)))
(A 1 B 4)
ACL2 !>(inversa (inversa '(1 2 3 4 5 6 7 8 9 10)))
(1 2 3 4 5 6 7 8 9 10)
```

El lenguaje de las fórmulas

- Conectivas lógicas
  - **`not`**, **`and`**, **`or`**, **`implies`**, **`iff`**, **`equal`**
- Cuantificadores
  - No hay funciones asociadas
  - Cuantificación universal implícita
  - Cuantificación existencial basada en funciones testigo (Skolem)
- Teoremas
  - **`defthm`**, **`thm`**

Ejemplo

- Lemas previos

```
(defthm concatena-inversa-nil
  (equal (concatena (inversa l) nil)
         (inversa l)))

(defthm concatena-asociativo
  (equal (concatena (concatena l1 l2) l3)
         (concatena l1 (concatena l2 l3))))
```

Ejemplo de sesión

- Entrada del usuario
  - Para ver el desarrollo de la demostración al completo (ACL2-8.0)

```
ACL2 !>:set-gag-mode nil
```

  - Escribimos la propiedad que queremos demostrar

```
ACL2 !>(defthm inversa-concatena
         (equal (inversa (concatena l1 l2))
                (concatena (inversa l2) (inversa l1))))
```

- Fórmulas en la "bolsa"

Fórmula 1

```
(equal (inversa (concatena a b))
       (concatena (inversa b) (inversa a)))
```

Ejemplo de sesión

- Procesando la fórmula 1

```
Name the formula above *1.

Perhaps we can prove *1 by induction.  Three induction schemes are
suggested by this conjecture.  Subsumption reduces that number to two.
However, one of these is flawed and so we are left with one viable
candidate.

We will induct according to a scheme suggested by (CONCATENA L1 L2).
This suggestion was produced using the :induction rules CONCATENA and
INVERSA.  If we let (:P L1 L2) denote *1 above then the induction scheme
we'll use is
(AND (IMPLIES (AND (NOT (ENDP L1)) (:P (CDR L1) L2))
              (:P L1 L2))
     (IMPLIES (ENDP L1) (:P L1 L2))).
This induction is justified by the same argument used to admit CONCATENA.
When applied to the goal at hand the above induction scheme produces
two nontautological subgoals.
```

Ejemplo de sesión

- Fórmulas en la "bolsa"

Fórmula *1/1

```
(IMPLIES (ENDP L1)
         (EQUAL (INVERSA (CONCATENA L1 L2))
                (CONCATENA (INVERSA L2) (INVERSA L1))))
```

Fórmula *1/2

```
(IMPLIES (AND (NOT (ENDP L1))
              (EQUAL (INVERSA (CONCATENA (CDR L1) L2))
                     (CONCATENA (INVERSA L2)
                                (INVERSA (CDR L1)))))
         (EQUAL (INVERSA (CONCATENA L1 L2))
                (CONCATENA (INVERSA L2) (INVERSA L1))))
```

Ejemplo de sesión

- Procesando la fórmula `*1/2`: Se aplica simplificación

```
Subgoal *1/2
(IMPLIES (AND (NOT (ENDP L1))
              (EQUAL (INVERSA (CONCATENA (CDR L1) L2))
                     (CONCATENA (INVERSA L2)
                                (INVERSA (CDR L1)))))
         (EQUAL (INVERSA (CONCATENA L1 L2))
                (CONCATENA (INVERSA L2) (INVERSA L1)))).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/2'
(IMPLIES (AND (CONSP L1)
              (EQUAL (INVERSA (CONCATENA (CDR L1) L2))
                     (CONCATENA (INVERSA L2)
                                (INVERSA (CDR L1)))))
         (EQUAL (INVERSA (CONCATENA L1 L2))
                (CONCATENA (INVERSA L2) (INVERSA L1)))).
```

Ejemplo de sesión

- Fórmulas en la "bolsa"

Fórmula `*1/1`

```
...
```

Fórmula `*1/2'`

```
(IMPLIES (AND (CONSP L1)
              (EQUAL (INVERSA (CONCATENA (CDR L1) L2))
                     (CONCATENA (INVERSA L2)
                                (INVERSA (CDR L1)))))
         (EQUAL (INVERSA (CONCATENA L1 L2))
                (CONCATENA (INVERSA L2) (INVERSA L1))))
```

Ejemplo de sesión

- Procesando la fórmula `*1/2'`: Se aplica simplificación

```
Subgoal *1/2'
(IMPLIES (AND (CONSP L1)
              (EQUAL (INVERSA (CONCATENA (CDR L1) L2))
                     (CONCATENA (INVERSA L2)
                                (INVERSA (CDR L1)))))
         (EQUAL (INVERSA (CONCATENA L1 L2))
                (CONCATENA (INVERSA L2) (INVERSA L1)))).

This simplifies, using the :definitions CONCATENA and INVERSA, primitive
type reasoning and the :rewrite rules CAR-CONS and CDR-CONS, to

Subgoal *1/2''
(IMPLIES (AND (CONSP L1)
              (EQUAL (INVERSA (CONCATENA (CDR L1) L2))
                     (CONCATENA (INVERSA L2)
                                (INVERSA (CDR L1)))))
         (EQUAL (CONCATENA (INVERSA (CONCATENA (CDR L1) L2))
                           (LIST (CAR L1)))
                (CONCATENA (INVERSA L2)
                           (CONCATENA (INVERSA (CDR L1))
                                      (LIST (CAR L1)))))).
```

Ejemplo de sesión

- Fórmulas en la "bolsa"

Fórmula `*1/1`

...

Fórmula `*1/2''`

```
(IMPLIES (AND (CONSP L1)
              (EQUAL (INVERSA (CONCATENA (CDR L1) L2))
                     (CONCATENA (INVERSA L2)
                                (INVERSA (CDR L1)))))
         (EQUAL (CONCATENA (INVERSA (CONCATENA (CDR L1) L2))
                           (LIST (CAR L1)))
                (CONCATENA (INVERSA L2)
                           (CONCATENA (INVERSA (CDR L1))
                                      (LIST (CAR L1))))))
```

Ejemplo de sesión

- Procesando la fórmula `*1/2''`: No es posible aplicar simplificación, se aplica eliminación de destructores

```
Subgoal *1/2''
(IMPLIES (AND (CONSP L1)
              (EQUAL (INVERSA (CONCATENA (CDR L1) L2))
                     (CONCATENA (INVERSA L2)
                                (INVERSA (CDR L1)))))
         (EQUAL (CONCATENA (INVERSA (CONCATENA (CDR L1) L2))
                           (LIST (CAR L1)))
                (CONCATENA (INVERSA L2)
                           (CONCATENA (INVERSA (CDR L1))
                                      (LIST (CAR L1)))))).

The destructor terms (CAR L1) and (CDR L1) can be eliminated by using
CAR-CDR-ELIM to replace L1 by (CONS L3 L4), (CAR L1) by L3 and (CDR L1)
by L4.  This produces the following goal.

Subgoal *1/2'''
(IMPLIES (AND (CONSP (CONS L3 L4))
              (EQUAL (INVERSA (CONCATENA L4 L2))
                     (CONCATENA (INVERSA L2) (INVERSA L4))))
         (EQUAL (CONCATENA (INVERSA (CONCATENA L4 L2))
                           (LIST L3))
                (CONCATENA (INVERSA L2)
                           (CONCATENA (INVERSA L4) (LIST L3))))).
```

Ejemplo de sesión

- Fórmulas en la "bolsa"

Fórmula `*1/1`

```
...
```

Fórmula `*1/2'''`

```
(IMPLIES (AND (CONSP (CONS L3 L4))
              (EQUAL (INVERSA (CONCATENA L4 L2))
                     (CONCATENA (INVERSA L2) (INVERSA L4))))
              (EQUAL (CONCATENA (INVERSA (CONCATENA L4 L2))
                                (LIST L3))
              (CONCATENA (INVERSA L2)
                         (CONCATENA (INVERSA L4)
                                    (LIST L3)))))
```

Ejemplo de sesión

- Procesando la fórmula `*1/2'''`: Se aplica simplificación

```
Subgoal *1/2'''
(IMPLIES (AND (CONSP (CONS L3 L4))
              (EQUAL (INVERSA (CONCATENA L4 L2))
                     (CONCATENA (INVERSA L2) (INVERSA L4))))
         (EQUAL (CONCATENA (INVERSA (CONCATENA L4 L2))
                           (LIST L3))
                (CONCATENA (INVERSA L2)
                           (CONCATENA (INVERSA L4) (LIST L3))))).

This simplifies, using primitive type reasoning, to

Subgoal *1/2'4'
(IMPLIES (EQUAL (INVERSA (CONCATENA L4 L2))
                (CONCATENA (INVERSA L2) (INVERSA L4)))
         (EQUAL (CONCATENA (INVERSA (CONCATENA L4 L2))
                           (LIST L3))
                (CONCATENA (INVERSA L2)
                           (CONCATENA (INVERSA L4) (LIST L3))))).
```

Ejemplo de sesión

- Fórmulas en la "bolsa"

Fórmula `*1/1`

...

Fórmula `*1/2'4'`

```
(IMPLIES (EQUAL (INVERSA (CONCATENA L4 L2))
                (CONCATENA (INVERSA L2) (INVERSA L4)))
         (EQUAL (CONCATENA (INVERSA (CONCATENA L4 L2))
                           (LIST L3))
                (CONCATENA (INVERSA L2)
                           (CONCATENA (INVERSA L4)
                                      (LIST L3)))))
```

Ejemplo de sesión

- Procesando la fórmula `*1/2'4'`: No se puede aplicar simplificación, ni eliminación de destructores; se usa una igualdad de las premisas

```
Subgoal *1/2'4'
(IMPLIES (EQUAL (INVERSA (CONCATENA L4 L2))
                (CONCATENA (INVERSA L2) (INVERSA L4)))
         (EQUAL (CONCATENA (INVERSA (CONCATENA L4 L2))
                           (LIST L3))
                (CONCATENA (INVERSA L2)
                           (CONCATENA (INVERSA L4) (LIST L3))))).

We now use the hypothesis by substituting
(CONCATENA (INVERSA L2) (INVERSA L4)) for (INVERSA (CONCATENA L4 L2))
and throwing away the hypothesis.  This produces

Subgoal *1/2'5'
(EQUAL (CONCATENA (CONCATENA (INVERSA L2) (INVERSA L4))
                  (LIST L3))
       (CONCATENA (INVERSA L2)
                  (CONCATENA (INVERSA L4) (LIST L3)))).
```

Ejemplo de sesión

- Fórmulas en la "bolsa"

Fórmula `*1/1`

...

Fórmula `*1/2'5'`

```
(EQUAL (CONCATENA (CONCATENA (INVERSA L2) (INVERSA L4))
                  (LIST L3))
       (CONCATENA (INVERSA L2)
                  (CONCATENA (INVERSA L4) (LIST L3))))
```

Ejemplo de sesión

- Procesando la fórmula `*1/2'5'`: Se aplica simplificación con la regla **CONCATENA-ASOCIATIVO**

```
Subgoal *1/2'5'
(EQUAL (CONCATENA (CONCATENA (INVERSA L2) (INVERSA L4))
                  (LIST L3))
       (CONCATENA (INVERSA L2)
                  (CONCATENA (INVERSA L4) (LIST L3)))).

But we reduce the conjecture to T, by the simple :rewrite rule
CONCATENA-ASOCIATIVO.
```

Ejemplo de sesión

- Fórmulas en la "bolsa"

Fórmula *1/1

```
(IMPLIES (ENDP A)
         (EQUAL (INVERSA (CONCATENA A B))
                (CONCATENA (INVERSA B) (INVERSA A))))
```

- Procesando la fórmula *1/1

```
Subgoal *1/1
(IMPLIES (ENDP L1)
         (EQUAL (INVERSA (CONCATENA L1 L2))
                (CONCATENA (INVERSA L2) (INVERSA L1)))).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/1'
(IMPLIES (NOT (CONSP L1))
         (EQUAL (INVERSA (CONCATENA L1 L2))
                (CONCATENA (INVERSA L2) (INVERSA L1)))).

But simplification reduces this to T, using the :definitions CONCATENA
and INVERSA, primitive type reasoning and the :rewrite rule
CONCATENA-INVERSA-NIL.
```

Ejemplo de sesión

- La "bolsa" queda vacía, por tanto la fórmula inicial es un teorema

```
That completes the proof of *1.

Q.E.D.

Summary
Form:  ( DEFTHM INVERSA-CONCATENA ...)
Rules: ((:DEFINITION CONCATENA)
        (:DEFINITION ENDP)
        (:DEFINITION INVERSA)
        (:DEFINITION NOT)
        (:ELIM CAR-CDR-ELIM)
        (:FAKE-RUNE-FOR-TYPE-SET NIL)
        (:INDUCTION CONCATENA)
        (:INDUCTION INVERSA)
        (:REWRITE CAR-CONS)
        (:REWRITE CDR-CONS)
        (:REWRITE CONCATENA-ASOCIATIVO)
        (:REWRITE CONCATENA-INVERSA-NIL))
Time:  0.01 seconds (prove: 0.01, print: 0.00, other: 0.00)
Prover steps counted:  1891
 INVERSA-CONCATENA
```

Bibliografía

- Kaufmann, M. and Manolios, P. and Moore, J S. *Computer–Aided Reasoning: An Approach*. (Kluwer Academic Publishers, 2000)
- Kaufmann, M. and Moore, J S. ACL2 *Version 8.5*. (`http://www.cs.utexas.edu/users/moore/acl2/`)
- Kaufmann, M. and Moore, J S. *A Flying Tour of* ACL2