Desarrollo de teorías computacionales

José A. Alonso Jiménez y Grupo de Lógica Computacional

Dpto. Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Antecedentes

- G. Frege (1879) *Conceptografía*

  Creo poder hacer muy clara la relación de mi conceptografía con el lenguaje común si la comparo con la que hay entre el microscopio y el ojo.

- J. McCarthy (1961) *A basis for a mathematical theory of computation*

  Instead of debugging a program, one should proved that its meets its specifications, and this proof should be checked by a computer program.

- Manifiesto QED (1993).

  QED is the very tentative title of a project to build a computer system that effectively represents all important mathematical knowledge and techniques.

Manifiesto QED

Nine reasons why the QED project should be undertaken

1. to help mathematicians cope with the explosion in mathematical knowledge
2. to help development of highly complex IT systems by facilitating the use of formal techniques
3. to help in mathematical education
4. to provide a cultural monument to "the fundamental reality of truth"
5. to help preserve mathematics from corruption
6. to help reduce the ‘noise level’ of published mathematics
7. to help make mathematics more coherent
8. to add to the body of explicitly formulated mathematics
9. to help improve the low level of self-consciousness in mathematics

El sistema ACL2

- **A C**omputacional **L**ogic for an **A**pplicative **C**ommon **L**isp.
- Desarrollado por Moore y Kaufmann.
- Sucesor de NQTHM de Boyer y Moore.
- Tres aspectos de ACL2:
  1. Lenguaje de programación.
  2. Lógica.
  3. Sistema de razonamiento automático.

ACL2 como lenguaje de programación

- Definición:

```
(defun concatena (x y)
  (if (endp x) y
      (cons (car x) (concatena (cdr x) y))))

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

- Evaluación:

```
ACL2 !> (concatena '(a b) '(c d e))
(A B C D E)
ACL2 !> (inversa '(a b c))
(C B A)
ACL2 !> (inversa (inversa '(a b c)))
(A B C)
```

ACL2 como lógica

- ACL2 es una lógica de primer orden con igualdad sin cuantificadores de la parte aplicativa de Common Lisp.
- Ejemplo de definiciones de la teoría de listas:

```
(defun true-listp (x)
  (if (consp x) (true-listp (cdr x))
                (eq x nil)))
```

- Ejemplo de axiomas de la teoría de lista:

```
(defaxiom car-cdr-elim
  (implies (consp x)
           (equal (cons (car x) (cdr x)) x)))

(defaxiom car-cons (equal (car (cons x y)) x))

(defaxiom cdr-cons (equal (cdr (cons x y)) y))
```

- Mediante `defun` se extiende la lógica de manera conservativa. Se necesitan condiciones de admisibilidad.

ACL2 como lógica: Admisibilidad

- Ejemplo 1:

```
ACL2 !> (defun f (x) y)

ACL2 Error in (DEFUN F ...): The body of F contains
a free occurrence of the variable symbol Y.
```

- Ejemplo 2:

```
ACL2 !> (defun f (x) (+ (f x) 1))

ACL2 Error in (DEFUN F ...): No :measure was
supplied with the definition of F. Our heuristics
for guessing one have not made any suggestions.
No argument of the function is tested along every
branch and occurs as a proper subterm at the same
argument position in every recursive call. You must
specify a :measure.
```

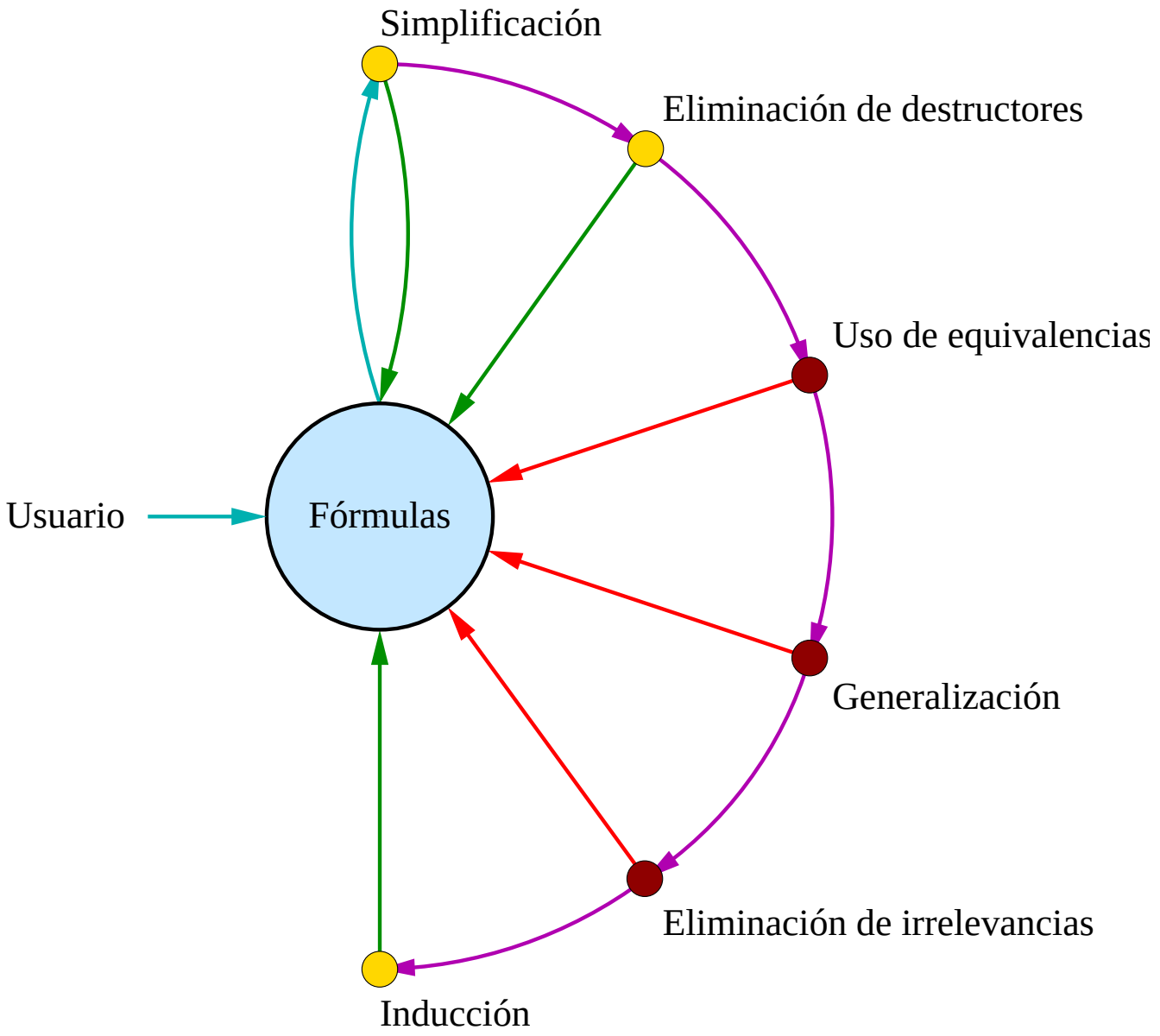
ACL2 como lógica: Admisibilidad

- Ejemplo 3:

```
ACL2 !> (defun inversa (x)
          (if (endp x)
              nil
              (concatena (inversa (cdr x))
                         (list (car x)))))

The admission of INVERSA is trivial, using the relation
E0-ORD-< (which is known to be well-founded on the
domain recognized by E0-ORDINALP) and the measure
(ACL2-COUNT X).  We observe that the type of INVERSA is
described by the theorem
(OR (CONSP (INVERSA X)) (EQUAL (INVERSA X) NIL)).
We used primitive type reasoning and the
:type-prescription rule CONCATENA.
```

ACL2 como SRA (sistema de razonamiento automático)



ACL2 como SRA: Inducción

```
ACL2 !> (defthm inversa-inversa
          (implies (true-listp x)
                   (equal (inversa (inversa x)) x)))

Name the formula above *1.

Perhaps we can prove *1 by induction. Two induction schemes
are suggested by this conjecture. These merge into one
derived induction scheme.
```

ACL2 como SRA: Inducción

```
We will induct according to a scheme suggested by (INVERSA
X). If we let (:P X) denote *1 above then the induction
scheme we'll use is
(AND (IMPLIES (AND (NOT (ENDP X)) (:P (CDR X)))
              (:P X))
     (IMPLIES (ENDP X) (:P X))).

This induction is justified by the same argument used to
admit INVERSA, namely, the measure (ACL2-COUNT X) is
decreasing according to the relation E0-ORD-< (which is known
to be well-founded on the domain recognized by E0-ORDINALP).
When applied to the goal at hand the above induction scheme
produces the following three nontautological subgoals.
```

ACL2 como SRA: Inducción

```
Subgoal *1/3
(IMPLIES (AND (NOT (ENDP X))
              (EQUAL (INVERSA (INVERSA (CDR X)))
                     (CDR X))
              (TRUE-LISTP X))
         (EQUAL (INVERSA (INVERSA X)) X)).

Subgoal *1/2
(IMPLIES (AND (NOT (ENDP X))
              (NOT (TRUE-LISTP (CDR X)))
              (TRUE-LISTP X))
         (EQUAL (INVERSA (INVERSA X)) X)).

Subgoal *1/1
(IMPLIES (AND (ENDP X) (TRUE-LISTP X))
         (EQUAL (INVERSA (INVERSA X)) X)).
```

ACL2 como SRA: Simplificación

```
Subgoal *1/3
(IMPLIES (AND (NOT (ENDP X))
              (EQUAL (INVERSA (INVERSA (CDR X)))
                     (CDR X))
              (TRUE-LISTP X))
         (EQUAL (INVERSA (INVERSA X)) X)).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/3'
(IMPLIES (AND (CONSP X)
              (EQUAL (INVERSA (INVERSA (CDR X)))
                     (CDR X))
              (TRUE-LISTP X))
         (EQUAL (INVERSA (INVERSA X)) X)).
```

ACL2 como SRA: Simplificación

```
Subgoal *1/3'
(IMPLIES (AND (CONSP X)
              (EQUAL (INVERSA (INVERSA (CDR X)))
                     (CDR X))
              (TRUE-LISTP X))
         (EQUAL (INVERSA (INVERSA X)) X)).

This simplifies, using the :definitions INVERSA and
TRUE-LISTP, to

Subgoal *1/3''
(IMPLIES (AND (CONSP X)
              (EQUAL (INVERSA (INVERSA (CDR X)))
                     (CDR X))
              (TRUE-LISTP (CDR X)))
         (EQUAL (INVERSA (CONCATENA (INVERSA (CDR X))
                                    (LIST (CAR X))))
                X)).
```

ACL2 como SRA: Eliminación de destructores

```
Subgoal *1/3''
(IMPLIES (AND (CONSP X)
              (EQUAL (INVERSA (INVERSA (CDR X))) (CDR X))
              (TRUE-LISTP (CDR X)))
         (EQUAL (INVERSA (CONCATENA (INVERSA (CDR X))
                                    (LIST (CAR X))))
                X)).
```

The destructor terms (CAR X) and (CDR X) can be eliminated by using CAR-CDR-ELIM to replace X by (CONS X1 X2), generalizing (CAR X) to X1 and (CDR X) to X2. This produces the following goal.

```
Subgoal *1/3'''
(IMPLIES (AND (CONSP (CONS X1 X2))
              (EQUAL (INVERSA (INVERSA X2)) X2)
              (TRUE-LISTP X2))
         (EQUAL (INVERSA (CONCATENA (INVERSA X2) (LIST X1)))
                (CONS X1 X2))).
```

ACL2 como SRA: Eliminación de destructores

```
Subgoal *1/3'''
(IMPLIES (AND (CONSP (CONS X1 X2))
              (EQUAL (INVERSA (INVERSA X2)) X2)
              (TRUE-LISTP X2))
         (EQUAL (INVERSA (CONCATENA (INVERSA X2) (LIST X1)))
                (CONS X1 X2))).

This simplifies, using primitive type reasoning and the
:type-prescription rule INVERSA, to

Subgoal *1/3'4'
(IMPLIES (AND (EQUAL (INVERSA (INVERSA X2)) X2)
              (TRUE-LISTP X2))
         (EQUAL (INVERSA (CONCATENA (INVERSA X2) (LIST X1)))
                (CONS X1 X2))).
```

ACL2 como SRA: Uso de equivalencias

```
Subgoal *1/3'4'
(IMPLIES (AND (EQUAL (INVERSA (INVERSA X2)) X2)
              (TRUE-LISTP X2))
         (EQUAL (INVERSA (CONCATENA (INVERSA X2) (LIST X1)))
                (CONS X1 X2))).
```

We now use the first hypothesis by cross-fertilizing (INVERSA (INVERSA X2)) for X2 and throwing away the hypothesis. This produces

```
Subgoal *1/3'5'
(IMPLIES (TRUE-LISTP X2)
         (EQUAL (INVERSA (CONCATENA (INVERSA X2) (LIST X1)))
                (CONS X1 (INVERSA (INVERSA X2))))).
```

ACL2 como SRA: Generalización

```
Subgoal *1/3'5'
(IMPLIES (TRUE-LISTP X2)
         (EQUAL (INVERSA (CONCATENA (INVERSA X2) (LIST X1)))
                (CONS X1 (INVERSA (INVERSA X2))))).

We generalize this conjecture, replacing (INVERSA X2) by IA.
This produces

Subgoal *1/3'6'
(IMPLIES (TRUE-LISTP X2)
         (EQUAL (INVERSA (CONCATENA IA (LIST X1)))
                (CONS X1 (INVERSA IA)))).
```

ACL2 como SRA: Eliminación de irrelevancias

```
Subgoal *1/3'6'
(IMPLIES (TRUE-LISTP X2)
         (EQUAL (INVERSA (CONCATENA IA (LIST X1)))
                (CONS X1 (INVERSA IA)))).
```

We suspect that the term (TRUE-LISTP X2) is irrelevant to the truth of this conjecture and throw it out. We will thus try to prove

```
Subgoal *1/3'7'
(EQUAL (INVERSA (CONCATENA IA (LIST X1)))
       (CONS X1 (INVERSA IA))).
```

Name the formula above *1.1.

ACL2 como SRA: Simplificación

```
Subgoal *1/2
(IMPLIES (AND (NOT (ENDP X))
              (NOT (TRUE-LISTP (CDR X)))
              (TRUE-LISTP X))
         (EQUAL (INVERSA (INVERSA X)) X)).

But we reduce the conjecture to T, by primitive type
reasoning.
```

ACL2 como SRA: Simplificación

```
Subgoal *1/1
(IMPLIES (AND (ENDP X) (TRUE-LISTP X))
         (EQUAL (INVERSA (INVERSA X)) X)).

By the simple :definition ENDP we reduce the conjecture to

Subgoal *1/1'
(IMPLIES (AND (NOT (CONSP X)) (TRUE-LISTP X))
         (EQUAL (INVERSA (INVERSA X)) X)).

But simplification reduces this to T, using the :definition
TRUE-LISTP, the :executable-counterparts of CONSP, EQUAL and
INVERSA and primitive type reasoning.
```

ACL2 como SRA: Inducción y simplificación

```
So we now return to *1.1, which is

(EQUAL (INVERSA (CONCATENA IA (LIST X1)))
       (CONS X1 (INVERSA IA))).

Perhaps we can prove *1.1 by induction.  ...

Subgoal *1.1/2
(IMPLIES (AND (NOT (ENDP IA))
              (EQUAL (INVERSA (CONCATENA (CDR IA) (LIST X1)))
                     (CONS X1 (INVERSA (CDR IA)))))
         (EQUAL (INVERSA (CONCATENA IA (LIST X1)))
                (CONS X1 (INVERSA IA)))).
Subgoal *1.1/1
(IMPLIES (ENDP IA)
         (EQUAL (INVERSA (CONCATENA IA (LIST X1)))
                (CONS X1 (INVERSA IA)))).

That completes the proofs of *1.1 and *1. Q.E.D.
```

ACL2 como SRA: Resumen

```
Summary
Form:  ( DEFTHM INVERSA-INVERSA ...)
Rules: ((:DEFINITION CONCATENA)
        (:DEFINITION ENDP)
        (:DEFINITION INVERSA)
        (:DEFINITION NOT)
        (:DEFINITION TRUE-LISTP)
        (:ELIM CAR-CDR-ELIM)
        (:EXECUTABLE-COUNTERPART CONSP)
        (:EXECUTABLE-COUNTERPART EQUAL)
        (:EXECUTABLE-COUNTERPART INVERSA)
        (:FAKE-RUNE-FOR-TYPE-SET NIL)
        (:REWRITE CAR-CONS)
        (:REWRITE CDR-CONS)
        (:TYPE-PRESCRIPTION INVERSA))
Warnings:  None
Time:  0.03 seconds (prove: 0.03, print: 0.00, other: 0.00)
```

ACL2 como SRA: Ampliación de la teoría

```
ACL2 !> (defthm inversa-inversa-inversa
          (implies (true-listp x)
                   (equal (inversa (inversa (inversa x)))
                          (inversa x))))

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

Q.E.D.
```

El papel del usuario

- Frecuentemente los primeros intentos de demostración de resultados no triviales fallan.
- Ello significa que el demostrador necesita demostrar lemas previos.
- Estos lemas se obtienen:
  - de una demostración preconcebida a mano o
  - del anális del fallo del intento de prueba.
- Por tanto, el papel del usuario es:
  - Formalizar la teoría en la lógica.
  - Implementar una estrategia de la prueba, mediante una sucesión de lemas.
- El resultado es un fichero con definiciones y teoremas
  - *Un libro* en la terminología de ACL2.
  - El libro puede certificarse y usarse en otros libros.

TRA en ACL2: Reducciones abstractas

Representación de $x \rightarrow y$ en ACL2.

```
(encapsulate
 ((q (x) boolean)
  (legal (x u) boolean)
  (reduce-one-step (x u) element))
  ...)
```

- `(q x)` se verifica si `x` pertenece al dominio de la relación.
- `(legal x op)` se verifica si se puede aplicar el operador `op` a `x`.
- `(reduce-one-step x op)` es el elemento obtenido aplicando el operador `op` a `x`.

TRA en ACL2: Pasos de prueba

- Representación de pasos de prueba:

```
(defstructure r-step direct operator elt1 elt2)
```

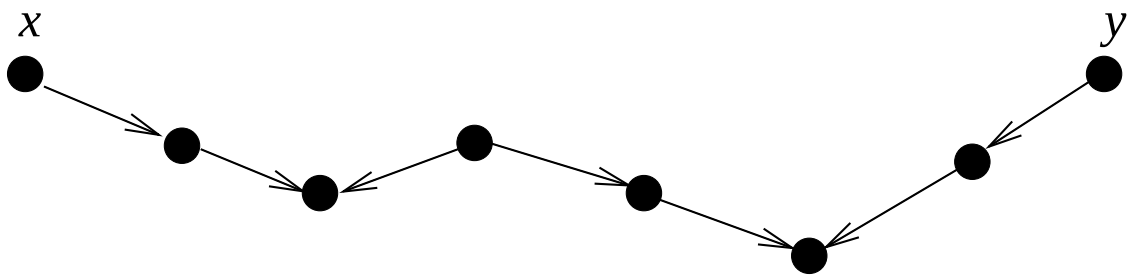
- Definición de pasos de prueba legales:

```
(defun proof-step-p (s)
  (let ((elt1 (elt1 s))
        (elt2 (elt2 s))
        (op (operator s))
        (direct (direct s)))
    (and (r-step-p s)
         (implies direct
                  (and (legal elt1 op)
                       (equal (reduce-one-step elt1 op)
                              elt2)))
         (implies (not direct)
                  (and (legal elt2 op)
                       (equal (reduce-one-step elt2 op)
                              elt1))))))
```

TRA en ACL2: Equivalencias

- Representación de $x \overset{*}{\leftrightarrow} y$ en ACL2:
  p es una prueba de la equivalencia de x e y:

```
(defun equiv-p (x y p)
   (if (endp p)
       (and (equal x y) (q x))
       (and (q x)
            (proof-step-p (car p))
            (equal x (elt1 (car p)))
            (equiv-p (elt2 (car p)) y (cdr p)))))
```



- Formas de pruebas:
  - Pico local: `(local-peak-p p)`: $y \leftarrow x \rightarrow z$
  - Valle: `steps-valley p)`: $y \overset{*}{\rightarrow} u \overset{*}{\leftarrow} z$

TRA en ACL2: Church–Rosser y normalización

- Church-Rosser (C–R): $y \overset{*}{\leftrightarrow} z \Longrightarrow y \overset{*}{\rightarrow} u \overset{*}{\leftarrow} z$
- Normalización (N): todo elemento tiene forma normal.

```
(encapsulate
 ((q ...) (legal ...) (reduce-one-step ...)
  (transform-to-valley ...) (proof-irreducible ...))
  .....
  (defthm Chuch-Rosser-property
   (let ((valley (transform-to-valley p)))
     (implies (equiv-p x y p)
              (and (steps-valley valley)
                   (equiv-p x y valley)))))
  .....
  (defthm normalizing
   (implies (q x)
            (let* ((p-x-y (proof-irreducible x))
                   (y (last-of-proof x p-x-y)))
              (and (equiv-p x y p-x-y)
                   (not (legal y op)))))))
```

TRA en ACL2: (C–R) y (N) implica decidibilidad de $\overset{*}{\leftrightarrow}$

- Forma normal de x:

```
(defun normal-form (x)
  (last-of-proof x (proof-irreducible x)))
```

- x e y tienen igual forma normal:

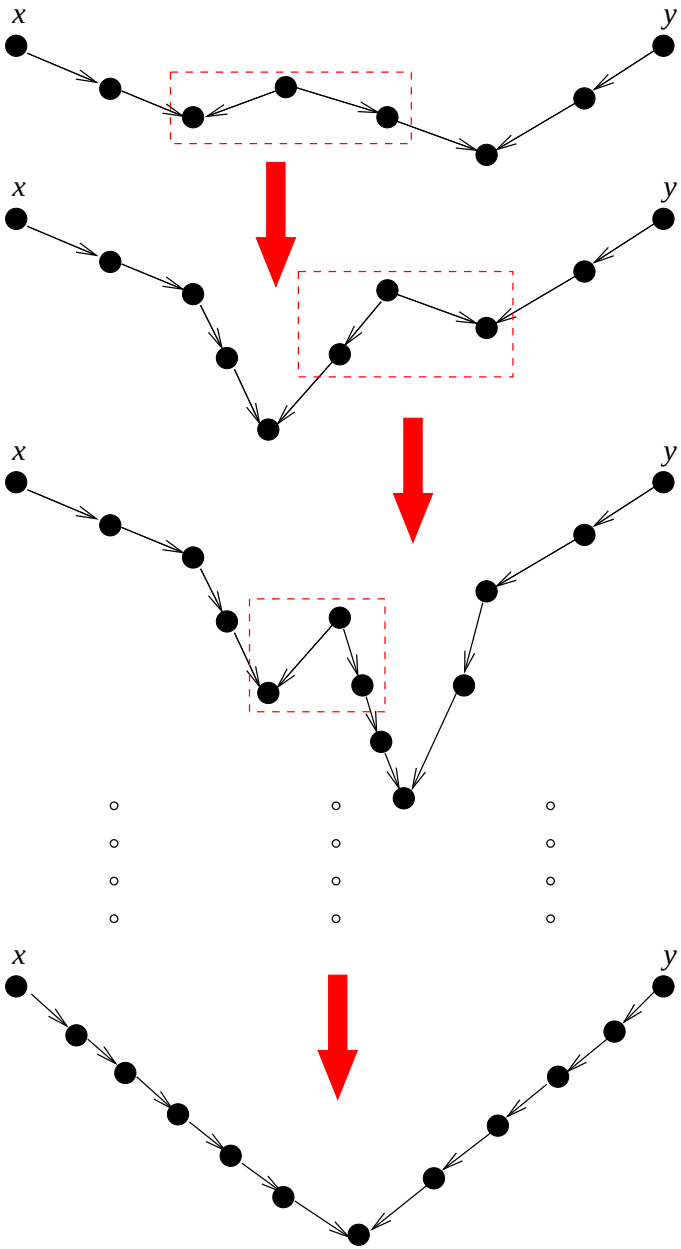
```
(defun r-equiv (x y)
  (equal (normal-form x) (normal-form y)))
```

- $x \overset{*}{\leftrightarrow} y$ syss tienen igual forma normal:

```
(defthm r-equiv-sound
  (implies (and (q x) (q y) (r-equiv x y))
           (equiv-p x y (make-proof-common-n-f x y))))

(defthm r-equiv-complete
  (implies (equiv-p x y p) (r-equiv x y)))
```

TRA en ACL2: Lema de Newmann (I)



TRA en ACL2: Lema de Newmann (II)

- Función auxiliar para expresar las hipótesis del lema de Newman: `rel` es una relación transitiva bien fundamentada en el dominio.

```
(encapsulate
 ((q ...) (legal ...) (reduce-one-step ...)
  (rel (x y) booleanp)
  (fn (x) e0-ordinalp)
  (transform-local-peak (x) proof))
  ...
  (defthm rel-well-founded-relation-on-q
   (and (implies (q x) (e0-ordinalp (fn x)))
        (implies (and (q x) (q y) (rel x y))
                 (e0-ord-< (fn x) (fn y))))
   :rule-classes (:well-founded-relation :rewrite))

  (defthm rel-transitive
     (implies (and (q x) (q y) (q z) (rel x y) (rel y z))
              (rel x z))
```

TRA en ACL2: Lema de Newmann (III)

- Hipótesis del lema de Newman: una reducción noetheriana y localmente confluente.

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

(defthm noetherian
  (implies (and (q x)
                (legal x u)
                (q (reduce-one-step x u)))
           (rel (reduce-one-step x u) x))))
```

TRA en ACL2: Lema de Newmann (IV)

- Conclusión del lema de Newman: $x \overset{*}{\leftrightarrow} y \Longrightarrow x \downarrow y$

```
(defthm Newman-lemma
  (let ((valley (transform-to-valley p)))
    (implies (equiv-p x y p)
             (and (steps-valley valley)
                  (equiv-p x y valley)))))
```

- Definición de `transform-to-valley`:

```
(defun transform-to-valley (p)
  (declare (xargs :measure (if (steps-q p)
                               (proof-measure p) nil)
                  :well-founded-relation mul-rel))
  (if (and (steps-q p) (exists-local-peak p))
      (transform-to-valley (replace-local-peak p))
      p))
```

- Comentarios:
  - Terminación de `transform-to-valley`.
  - Esquema de inducción para probar `Newman-lemma`.

Teorías ecuacionales en ACL2

- Representación de la reducción $t_1 \rightarrow_E t_2$
  - dominio de definición:
    `(term-s-p x)`
  - operadores ecuacionales:
    `(defstructure eq-operator rule pos matching)`
  - aplicabilidad de un operador:

```
(defun eq-legal (term op E)
  (let ((pos (op-pos op)) (rule (op-rule op))
        (sigma (op-match op)))
    (and (eq-operator-p op) (member rule E)
         (substitution-s-p sigma)
         (position-p pos term)
         (equal (instance (lhs rule) sigma)
                (occurrence term pos)))))
```

Teorías ecuacionales en ACL2

- Representación de la reducción $t_1 \rightarrow_E t_2$ (Cont.)
  - Un paso de reducción ecuacional:

```
(defun eq-reduce-one-step (term op)
  (replace-term term
                (op-pos op)
                (instance (rhs (op-rule op))
                          (op-match op))))
```

- La relación de equivalencia ecuacional (teoría ecuacional de E):

```
(defun eq-equiv-s-p (t1 t2 p E)
  (if (endp p)
      (and (equal t1 t2) (term-s-p t1))
      (and (term-s-p t1)
           (eq-proof-step-p (car p) E)
           (equal t1 (elt1 (car p)))
           (eq-equiv-s-p (elt2 (car p)) t2 (cdr p) E))))
```

Teorema de pares críticos

- Teorema de pares críticos de Knuth y Bendix: $\rightarrow_E$ es localmente confluente si para cualquier $(u, v) \in pc(E)$, se tiene $u \downarrow_E v$.
- Hipótesis del teorema en ACL2:

```
(encapsulate
 ((EKB () system-with-joinable-critical-pairs)
  (transform-critical-pair (l1 r1 p l2 r2) valley-proof))
 ....
 (defthm EKB-joinable-critical-pairs
   (implies (and (member (make-equation l1 r1) (EKB))
                 (member (make-equation l2 r2) (EKB))
                 (position-p p l1)
                 (not (variable-p (occurrence l1 p))))
            (let ((cp-r (cp-r l1 r1 p l2 r2))
                  (valley (transform-critical-pair l1 r1 p l2
              (implies cp-r
                       (and (eq-equiv-s-p (lhs cp-r)
                                          (rhs cp-r)
                                          valley (EKB))
                            (steps-valley valley)))))))
```

Teorema de pares críticos

- Conclusión del teorema en ACL2: `(EKB)` es localmente confluente: para toda prueba que sea un pico local, *existe* una prueba valle equivalente

```
(defthm knuth-bendix-theorem
  (implies
    (and (eq-equiv-s-p t1 t2 p (EKB))
         (local-peak-p p))
    (and (eq-equiv-s-p t1 t2
                       (transform-eq-local-peak p)
                       (EKB))
         (steps-valley (transform-eq-local-peak p)))))
```

- La definición de `transform-eq-local-peak`:
  - distinción de casos basada en la demostración a mano.
  - 200 líneas de código
  - demostración compleja
  - Buen ejemplo de reutilización de propiedades.