ACL2: Interacción con el sistema

Francisco J. Martín Mateos
José L. Ruiz Reina

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

Algoritmos de ordenación

- Algoritmo de inserción ordenada

```
(defun inserta (a x)
  (if (consp x)
      (if (<= a (car x))
          (cons a x)
          (cons (car x) (inserta a (cdr x))))
      (list a)))

(defun isort (x)
  (if (consp x)
      (inserta (car x) (isort (cdr x)))
      nil))
```

Algoritmos de ordenación

- Algoritmo de ordenación por mezclas

```
(defun trozo-i (l)
  (cond ((endp l) nil)
        ((endp (cdr l)) nil)
        (t (cons (car l) (trozo-i (cddr l))))))

(defun trozo-p (l)
  (cond ((endp l) nil)
        ((endp (cdr l)) l)
        (t (cons (cadr l) (trozo-p (cddr l))))))
```

Algoritmos de ordenación

- Algoritmo de ordenación por mezclas

```
(defun mezcla (l1 l2)
  (cond ((endp l1) l2)
        ((endp l2) l1)
        ((< (car l1) (car l2))
         (cons (car l1) (mezcla (cdr l1) l2)))
        (t (cons (car l2) (mezcla l1 (cdr l2))))))

(defun msort (l)
  (cond ((endp l) nil)
        ((endp (cdr l)) (list (car l)))
        (t (mezcla (msort (trozo-i l))
                   (msort (trozo-p l))))))
```

Algoritmos de ordenación

- Algoritmo de ordenación rápida

```
(defun qsort-< (x l)
  (cond ((endp l) nil)
        ((< (car l) x)
         (cons (car l) (qsort-< x (cdr l))))
        (t (qsort-< x (cdr l)))))

(defun qsort-no-< (x l)
  (cond ((endp l) nil)
        ((< (car l) x)
         (qsort-no-< x (cdr l)))
        (t (cons (car l) (qsort-no-< x (cdr l))))))
```

Algoritmos de ordenación

- Algoritmo de ordenación rápida

```
(defun qsort (l)
  (if (endp l)
      nil
      (append (qsort (qsort-< (car l) (cdr l)))
              (list (car l))
              (qsort (qsort-no-< (car l) (cdr l))))))
```

Propiedades de los algoritmos de ordenación

- El resultado es una lista ordenada de menor a mayor

```
(defun ordenada (l)
  (cond ((endp l) (equal l nil))
        (t (or (equal (cdr l) nil)
               (and (<= (first l) (second l))
                    (ordenada (cdr l)))))))
```

- Propiedades a demostrar
  - `(ordenada (isort l))`
  - `(ordenada (msort l))`
  - `(ordenada (qsort l))`

Propiedades de los algoritmos de ordenación

- El resultado tiene los mismos elementos que el original (permutación)

```
(defun borra-uno (x l)
  (cond ((endp l) l)
        ((equal x (car l)) (cdr l))
        (t (cons (car l) (borra-uno x (cdr l))))))

(defun perm (l1 l2)
  (cond ((endp l1) (endp l2))
        ((member (car l1) l2)
         (perm (cdr l1) (borra-uno (car l1) l2)))
        (t nil)))
```

- Propiedades a demostrar
  - `(perm (isort l) l)`
  - `(perm (msort l) l)`
  - `(perm (qsort l) l)`

Propiedades de `perm`: reflexividad

- Reflexividad de `perm`

```
(defthm perm-reflexiva
  (perm l l))
```

  - El resultado es probado directamente por el sistema

Propiedades de `perm`: transitividad

- Transitividad de `perm`

```
(defthm perm-transitiva
  (implies (and (perm l1 l2)
                (perm l2 l3))
           (perm l1 l3)))
```

  - El sistema no es capaz de demostrar el resultado
  - Buscamos en la prueba fallida un posible resultado previo antes de un paso de generalización

Propiedades de `perm`: transitividad

- Extracto de la prueba fallida

```
Subgoal *1/3'4'
(IMPLIES (AND (MEMBER-EQUAL L4 L3)
              (NOT (PERM (BORRA-UNO L4 L2)
                         (BORRA-UNO L4 L3)))
              (MEMBER-EQUAL L4 L2)
              (PERM L5 (BORRA-UNO L4 L2))
              (PERM L2 L3))
         (PERM L5 (BORRA-UNO L4 L3))).

We generalize this conjecture ...
```

Propiedades de `perm`: transitividad

- Comportamiento de `perm` con respecto a `borra-uno`

```
(defthm perm-borra-uno-borra-uno
  (implies (perm l1 l2)
           (perm (borra-uno x l1) (borra-uno x l2))))
```

  - El sistema no es capaz de demostrar el resultado
  - Buscamos en la prueba fallida un posible resultado previo antes de un paso de generalización

Propiedades de `perm`: transitividad

- Extracto de la prueba fallida

```
Subgoal *1/4.2''
(IMPLIES (AND (NOT (EQUAL X L3))
              (PERM (BORRA-UNO X L4)
                    (BORRA-UNO X (BORRA-UNO L3 L2)))
              (MEMBER-EQUAL L3 L2)
              (PERM L4 (BORRA-UNO L3 L2)))
         (MEMBER-EQUAL L3 (BORRA-UNO X L2))).
```

Propiedades de **perm**: transitividad

- Comportamiento de **member** con respecto a **borra-uno** (I)

```
(defthm member-borra-uno-1
  (implies (and (not (equal x y))
                (member x l))
           (member x (borra-uno y l))))
```

  - El resultado es probado directamente por el sistema
  - El resultado **perm-borra-uno-borra-uno** todavía no es demostrado por el sistema
  - Buscamos en la prueba fallida un posible resultado previo antes de un paso de generalización

Propiedades de `perm`: transitividad

- Extracto de la prueba fallida de `perm-borra-uno-borra-uno`

```
Subgoal *1/4'4'
(IMPLIES (AND (NOT (EQUAL X L3))
              (PERM (BORRA-UNO X L4)
                    (BORRA-UNO X (BORRA-UNO L3 L2)))
              (MEMBER-EQUAL L3 L2)
              (PERM L4 (BORRA-UNO L3 L2)))
         (PERM (BORRA-UNO X L4)
               (BORRA-UNO L3 (BORRA-UNO X L2)))).
```

Propiedades de **perm**: transitividad

- La función **borra-uno** es conmutativa

```
(defthm borra-uno-borra-uno
  (equal (borra-uno x (borra-uno y l))
         (borra-uno y (borra-uno x l))))
```

  - El resultado es probado directamente por el sistema
  - El resultado **perm-borra-uno-borra-uno** es demostrado por el sistema
  - El resultado **perm-transitiva** todavía no es demostrado por el sistema
  - Buscamos en la prueba fallida un posible resultado previo antes de un paso de generalización

Propiedades de `perm`: transitividad

- Extracto de la prueba fallida de `perm-transitiva`

```
Subgoal *1.1/4'4'
(IMPLIES (AND (MEMBER-EQUAL L6 L3)
              (NOT (PERM L5 (BORRA-UNO L4 L7)))
              (NOT (MEMBER-EQUAL L4 L3))
              (MEMBER-EQUAL L4 L7)
              (NOT (EQUAL L4 L6))
              (PERM L5 (CONS L6 (BORRA-UNO L4 L7))))
         (NOT (PERM L7 (BORRA-UNO L6 L3)))).
```

Propiedades de `perm`: transitividad

- Comportamiento de `member` con respecto a `perm`

```
(defthm perm-member
  (implies (and (not (member x l1))
                (member x l2))
           (not (perm l2 l1))))
```

  - El sistema no es capaz de demostrar el resultado
  - Buscamos en la prueba fallida un posible resultado previo antes de un paso de generalización

Propiedades de `perm`: transitividad

- Extracto de la prueba fallida

```
Subgoal *1/2'4'
(IMPLIES (AND (MEMBER-EQUAL L3 L1)
              (MEMBER-EQUAL X (BORRA-UNO L3 L1))
              (NOT (MEMBER-EQUAL X L1))
              (MEMBER-EQUAL X L4))
         (NOT (PERM L4 (BORRA-UNO L3 L1)))).
```

Propiedades de **perm**: transitividad

- Comportamiento de **member** con respecto a **borra-uno** (II)

```
(defthm member-borra-uno-2
  (implies (member x (borra-uno y l))
           (member x l)))
```

  - El resultado es probado directamente por el sistema
  - El resultado **perm-member** es demostrado por el sistema
  - El resultado **perm-transitiva** es demostrado por el sistema

Propiedades de `perm`: simetría

- Simetría de `perm`

```
(defthm perm-simetrica
  (implies (perm l1 l2)
           (perm l2 l1)))
```

  - El sistema no es capaz de demostrar el resultado
  - Buscamos en la prueba fallida un posible resultado previo antes de un paso de generalización

Propiedades de `perm`: simetría

- Extracto de la prueba fallida

```
Subgoal *1/3'4'
(IMPLIES (AND (MEMBER-EQUAL L3 L2)
              (PERM (BORRA-UNO L3 L2) L4)
              (PERM L4 (BORRA-UNO L3 L2)))
         (PERM L2 (CONS L3 L4))).
```

  - Algunas veces el resultado inmediatamente anterior a un paso de generalización no es el más adecuado
  - El término `(CONS L3 L4)` imposibilita el uso del esquema de inducción sugerido por la definición de `PERM`

Propiedades de `perm`: simetría

- Un paso de eliminación de destructores introdujo el término `(CONS L3 L4)`
- El subobjetivo anterior a este paso nos proporciona la misma información, esta vez sin el término `(CONS L3 L4)`

```
Subgoal *1/3''
(IMPLIES (AND (CONSP L1)
              (MEMBER-EQUAL (CAR L1) L2)
              (PERM (BORRA-UNO (CAR L1) L2) (CDR L1))
              (PERM (CDR L1) (BORRA-UNO (CAR L1) L2)))
         (PERM L2 L1)).
```

Propiedades de **perm**: simetría

- Comportamiento de **perm** con respecto a **borra-uno**, **car** y **cdr**

```
(defthm perm-borra-uno-car-cdr
  (implies (and (consp l1)
                (member (car l1) l2)
                (perm (borra-uno (car l1) l2)
                      (cdr l1)))
           (perm l2 l1)))
```

  - El resultado es probado directamente por el sistema
  - El resultado **perm-simetrica** es demostrado por el sistema

Propiedades de **perm**: equivalencia

- **perm** es una relación de equivalencia

```
(defequiv perm)
```

  - El resultado es probado directamente por el sistema gracias a las propiedades **perm-reflexiva**, **perm-transitiva** y **perm-simetrica**

Propiedades de `perm`: congruencias

- Congruencia de `borra-uno` con respecto a `perm`

```
(defthm perm-borra-uno-borra-uno-congruencia
  (implies (perm l1 l2)
           (perm (borra-uno x l1) (borra-uno x l2)))
  :rule-classes :congruence)
```

  - El resultado es probado directamente por el sistema gracias a la propiedad `perm-borra-uno-borra-uno`

Propiedades de `perm`: congruencias

- Congruencias de `append` con respecto a `perm`

```
(defthm append-perm-1
  (implies (perm l1 l2)
           (perm (append l1 l3) (append l2 l3)))
  :rule-classes :congruence)

(defthm append-perm-2
  (implies (perm l1 l2)
           (perm (append l3 l1) (append l3 l2)))
  :rule-classes :congruence)
```

  - El sistema no es capaz de demostrar el resultado
  - Buscamos en la prueba fallida un posible resultado previo antes de un paso de generalización

Propiedades de `perm`: congruencias

- Extracto de la prueba fallida

```
Subgoal *1/3.2''
(IMPLIES (AND (PERM (APPEND L5 L3)
                    (APPEND (BORRA-UNO L4 L2) L3))
              (MEMBER-EQUAL L4 L2)
              (PERM L5 (BORRA-UNO L4 L2)))
         (MEMBER-EQUAL L4 (APPEND L2 L3))).
```

Propiedades de **perm**: congruencias

- Comportamiento de **member** con respecto a **append**

```
(defthm member-append
  (implies (member (car l1) l2)
           (member (car l1) (append l2 l3))))
```

  - El resultado es probado directamente por el sistema
  - Los resultados **append-perm-1** y **append-perm-2** son demostrados por el sistema

Algoritmo de inserción ordenada

- Definición del algoritmo

```
(defun inserta (a x)
  (if (consp x)
      (if (<= a (car x))
          (cons a x)
          (cons (car x) (inserta a (cdr x))))
      (list a)))

(defun isort (x)
  (if (consp x)
      (inserta (car x) (isort (cdr x)))
      nil))
```

  - El sistema admite las definiciones

Algoritmo de inserción ordenada

- Propiedades del algoritmo

```
(defthm ordenada-isort
  (ordenada (isort l)))

(defthm perm-isort
  (perm (isort l) l))
```

  - Los resultados son probados directamente por el sistema

Algoritmo de ordenación por mezclas

- Definición del algoritmo

```
(defun trozo-i (l)
  (cond ((endp l) nil)
        ((endp (cdr l)) l)
        (t (cons (car l) (trozo-i (cddr l))))))

(defun trozo-p (l)
  (cond ((endp l) nil)
        ((endp (cdr l)) nil)
        (t (cons (cadr l) (trozo-p (cddr l))))))
```

  - El sistema admite las definiciones

Algoritmo de ordenación por mezclas

- Definición del algoritmo

```
(defun mezcla (l1 l2)
  (cond ((endp l1) l2)
        ((endp l2) l1)
        ((< (car l1) (car l2))
         (cons (car l1) (mezcla (cdr l1) l2)))
        (t (cons (car l2) (mezcla l1 (cdr l2))))))
```

  - El sistema no es capaz de deducir una medida de terminación adecuada para admitir la definición de **`mezcla`**
  - Una medida de terminación para la función **`mezcla`** es

    `(+ (len l1) (len l2))`

Algoritmo de ordenación por mezclas

- Definición del algoritmo

```
(defun mezcla (l1 l2)
  (declare (xargs :measure (+ (len l1) (len l2))))
  (cond ((endp l1) l2)
        ((endp l2) l1)
        ((< (car l1) (car l2))
         (cons (car l1) (mezcla (cdr l1) l2)))
        (t (cons (car l2) (mezcla l1 (cdr l2))))))
```

  - El sistema admite la definición de **`mezcla`** gracias a la medida de terminación proporcionada

Algoritmo de ordenación por mezclas

- Una medida de terminación para la función **msort** es **(len l)**
- Definición del algoritmo

```
(defun msort (l)
  (declare (xargs :measure (len l)))
  (cond ((endp l) nil)
        ((endp (cdr l)) (list (car l)))
        (t (mezcla (msort (trozo-i l))
                   (msort (trozo-p l))))))
```

  - El sistema no admite la definición de **msort**
  - Examinamos el intento de demostración de las conjeturas de terminación

Algoritmo de ordenación por mezclas

- Extracto de la prueba fallida

```
Subgoal *2/2'''
(IMPLIES (< (LEN (TROZO-I (CONS L1 L4)))
            (+ 1 (LEN L4)))
         (< (+ 1 (LEN (TROZO-I L4)))
            (+ 2 (LEN L4)))).
```

  - En ACL2-3.6 el subobjetivo es distinto, en lugar de `(+ 2 (LEN L4))` aparece `(+ 1 1 (LEN L4))`

Algoritmo de ordenación por mezclas

- Propiedades de la longitud de **trozo-i** (y **trozo-p**)

```
(defthm len-trozo-i
  (< (+ 1 (len (trozo-i l)))
     (+ 2 (len l))))

(defthm len-trozo-p
  (< (+ 1 (len (trozo-p l)))
     (+ 2 (len l))))
```

  - Los resultados son demostrados directamente por el sistema
  - La definición de **msort** es admitida por el sistema

Algoritmo de ordenación por mezclas

- Propiedades del algoritmo

```
(defthm ordenada-msort
  (ordenada (msort l)))
```

  - El resultado es probado directamente por el sistema

Algoritmo de ordenación por mezclas

- Propiedades del algoritmo

```
(defthm perm-msort
  (perm (msort l) l))
```

  - El sistema no es capaz de demostrar el resultado
  - Buscamos en la prueba fallida un posible resultado previo antes de un paso de generalización

Algoritmo de ordenación por mezclas

- Extracto de la prueba fallida

```
Subgoal *1/3'4'
(IMPLIES (AND (CONSP L2)
              (PERM (MSORT (TROZO-I (CONS L1 L2)))
                    (TROZO-I (CONS L1 L2)))
              (PERM (MSORT (TROZO-P (CONS L1 L2)))
                    (TROZO-P (CONS L1 L2))))
         (PERM (MEZCLA (MSORT (TROZO-I (CONS L1 L2)))
                       (MSORT (TROZO-P (CONS L1 L2))))
               (CONS L1 L2))).
```

  - El término `(CONS L1 L2)` dificulta la visualización de este subobjetivo

Algoritmo de ordenación por mezclas

- Un paso de eliminación de destructores introdujo el término `(CONS L1 L2)`
- El subobjetivo anterior a este paso nos proporciona la misma información, esta vez sin el término `(CONS L1 L2)`

```
Subgoal *1/3''
(IMPLIES (AND (CONSP L)
              (CONSP (CDR L))
              (PERM (MSORT (TROZO-I L)) (TROZO-I L))
              (PERM (MSORT (TROZO-P L)) (TROZO-P L)))
         (PERM (MEZCLA (MSORT (TROZO-I L))
                       (MSORT (TROZO-P L)))
               L)).
```

Algoritmo de ordenación por mezclas

- Una planificación para demostrar este subobjetivo es la siguiente:
  - Demostrar que **`(append (trozo-i l) (trozo-p l))`** y **`l`** tienen los mismos elementos
  - Demostrar que **`(mezcla l1 l2)`** y **`(append l1 l2)`** tienen los mismos elementos

Algoritmo de ordenación por mezclas

- `(append (trozo-i l) (trozo-p l))` y `l` tienen los mismos elementos

```
(defthm perm-append-trozo-i-p
  (perm (append (trozo-i l) (trozo-p l)) l))
```

  - El sistema no es capaz de demostrar el resultado
  - Buscamos en la prueba fallida de **`perm-append-trozo-i-p`** un posible resultado previo antes de un paso de generalización

Algoritmo de ordenación por mezclas

- Extracto de la prueba fallida

```
Subgoal *1/3'4'
(IMPLIES (PERM (APPEND (TROZO-I L4) (TROZO-P L4))
               L4)
         (PERM (APPEND (TROZO-I L4)
                       (CONS L3 (TROZO-P L4)))
               (CONS L3 L4))).
```

Algoritmo de ordenación por mezclas

- Comportamiento de **perm** con respecto a **append** y **cons**

```
(defthm perm-append-cons
  (perm (append l1 (cons x l2))
        (cons x (append l1 l2))))
```

  - El resultado es demostrado directamente por el sistema
  - El resultado **perm-append-trozo-i-p** es demostrado directamente por el sistema

Algoritmo de ordenación por mezclas

- `(mezcla l1 l2)` y `(append l1 l2)` tienen los mismos elementos

```
(defthm perm-mezcla-append
  (perm (mezcla l1 l2)
        (append l1 l2)))
```

  - El sistema no es capaz de demostrar el resultado
  - El sistema escoge el esquema de inducción sugerido por `(append l1 l2)`, que no distingue los casos base de `(mezcla l1 l2)`
  - El esquema de inducción sugerido por `(mezcla l1 l2)` es más adecuado

Algoritmo de ordenación por mezclas

- `(mezcla l1 l2)` y `(append l1 l2)` tienen los mismos elementos
- Indicamos el esquema de inducción sugerido por `(mezcla l1 l2)`

```
(defthm perm-mezcla-append
  (perm (mezcla l1 l2)
        (append l1 l2))
  :hints (("Goal" :induct (mezcla l1 l2))))
```

  - El sistema no es capaz de demostrar el resultado
  - Buscamos en la prueba fallida de `perm-mezcla-append` un posible resultado previo antes de un paso de generalización

Algoritmo de ordenación por mezclas

- Extracto de la prueba fallida

```
Subgoal *1.1/2.1''
(IMPLIES (AND (NOT (PERM (MEZCLA (CONS L2 L7) L4)
                         (CONS L2 (APPEND L7 L4))))
              (<= L3 L2)
              (PERM (MEZCLA (LIST* L2 L6 L7) L4)
                    (LIST* L2 L6 (APPEND L7 L4)))
              (NOT (EQUAL L3 L2))
              (NOT (EQUAL L3 L6)))
         (MEMBER L3 (APPEND L7 (CONS L3 L4)))).
```

Algoritmo de ordenación por mezclas

- Comportamiento de **member** con respecto a **append** y **cons**

```
(defthm member-append-cons
  (member x (append l1 (cons x l2))))
```

  - El resultado es demostrado directamente por el sistema
  - El resultado **perm-mezcla-append** es demostrado directamente por el sistema
  - El resultado **perm-msort** es demostrado directamente por el sistema

Algoritmo de ordenación rápida

- Definición del algoritmo

```
(defun qsort-< (x l)
  (cond ((endp l) nil)
        ((< (car l) x)
         (cons (car l) (qsort-< x (cdr l))))
        (t (qsort-< x (cdr l)))))

(defun qsort-no-< (x l)
  (cond ((endp l) nil)
        ((< (car l) x)
         (qsort-no-< x (cdr l)))
        (t (cons (car l) (qsort-no-< x (cdr l))))))
```

  - El sistema admite las definiciones

Algoritmo de ordenación rápida

- Definición del algoritmo

```
(defun qsort (l)
  (cond ((endp l) nil)
        (t (append
              (qsort (qsort-< (car l) (cdr l)))
              (list (car l))
              (qsort (qsort-no-< (car l) (cdr l)))))))
```

Algoritmo de ordenación rápida

- Propiedades del algoritmo

```
(defthm perm-qsort
  (perm (qsort l) l))
```

  - El sistema no es capaz de demostrar el resultado
  - Buscamos en la prueba fallida de **perm-mezcla-append** un posible resultado previo antes de un paso de generalización

Algoritmo de ordenación rápida

- Extracto de la prueba fallida

```
Subgoal *1/2'4'
(IMPLIES (AND (PERM (QSORT (QSORT-< L1 L2))
                    (QSORT-< L1 L2))
              (PERM (QSORT (QSORT-NO-< L1 L2))
                    (QSORT-NO-< L1 L2)))
         (PERM (APPEND (QSORT-< L1 L2)
                       (QSORT-NO-< L1 L2))
               L2)).
```

Algoritmo de ordenación rápida

- Comportamiento de **perm** con respecto a **append**, **qsort-<** y **qsort-no-<**

```
(defthm perm-append-qsort-<-no-<
  (perm (append (qsort-< x l)
                (qsort-no-< x l))
        l))
```

  - El resultado es demostrado directamente por el sistema
  - El resultado **perm-qsort** es demostrado directamente por el sistema

Algoritmo de ordenación rápida

- Propiedades del algoritmo

```
(defthm ordenada-qsort
  (ordenada (qsort l)))
```

  - El sistema no es capaz de demostrar el resultado
  - Buscamos en la prueba fallida de **`ordenada-qsort`** un posible resultado previo antes de un paso de generalización

Algoritmo de ordenación rápida

- Extracto de la prueba fallida

```
Subgoal *1/2'4'
(IMPLIES (AND (ORDENADA (QSORT (QSORT-< L1 L2)))
              (ORDENADA (QSORT (QSORT-NO-< L1 L2))))
         (ORDENADA
            (APPEND (QSORT (QSORT-< L1 L2))
                    (CONS L1 (QSORT (QSORT-NO-< L1 L2))))))).
```

  - ¿En que circustancias es cierto `(ordenada (append l1 (cons x l2)))`?

Algoritmo de ordenación rápida

- Una planificación para demostrar este subobjetivo es la siguiente:
  - Demostrar que si `(ordenada l1)`, `(ordenada l2)` y el elemento `e` es mayor o igual que los elementos de `l1` y menor o igual que los elementos de `l2` entonces

    `(ordenada (append l1 (cons e l2)))`
  - Definir "un elemento es mayor o igual que los elementos de una lista"
  - Definir "un elemento es menor o igual que los elementos de una lista"
  - Demostrar que todo elemento `e` es mayor o igual que los elementos de la lista `(qsort (qsort-< e l))`
  - Demostrar que todo elemento `e` es menor o igual que los elementos de la lista `(qsort (qsort-no-< e l))`

Algoritmo de ordenación rápida

- Definiciones auxiliares
  - El elemento `x` es mayor o igual que los elementos de la lista `l`

```
(defun lista-<= (l x)
  (cond ((endp l) t)
        (t (and (<= (car l) x)
                (lista-<= (cdr l) x)))))
```

  - El elemento `x` es menor o igual que los elementos de la lista `l`

```
(defun <=-lista (x l)
  (cond ((endp l) t)
        (t (and (<= x (car l))
                (<=-lista x (cdr l))))))
```

Algoritmo de ordenación rápida

- Caracterización de **ordenada** con respecto a **append**

```
(defthm ordenada-append
  (implies (and (ordenada l1)
                (ordenada l2)
                (lista-<= l1 x)
                (<=-lista x l2))
           (ordenada (append l1 (cons x l2)))))
```

  - El resultado es demostrado directamente por el sistema

Algoritmo de ordenación rápida

- Todo elemento `e` es mayor o igual que los elementos de la lista

  `(qsort (qsort-< e l))`

```
(defthm lista-<=-qsort-qsort-<
  (lista-<= (qsort (qsort-< x l)) x))
```

  - La aparición consecutiva de las funciones `qsort` y `qsort-<` dificulta la prueba de este resultado

Algoritmo de ordenación rápida

- Una planificación para demostrar este subobjetivo consiste en usar congruencias con respecto a **`perm`**
  - **`(qsort (qsort-< x l))`** es una permutación de **`(qsort-< x l)`** (propiedad **`perm-qsort`**)
  - **`lista-<=`** es congruente con respecto a **`perm`** en su primer argumento

Algoritmo de ordenación rápida

- **`lista-<=`** es congruente con respecto a **`perm`** en su primer argumento

```
(defthm lista-<=-perm
  (implies (perm l1 l2)
           (equal (lista-<= l1 x)
                  (lista-<= l2 x)))
  :rule-classes :congruence)
```

  - El resultado es demostrado directamente por el sistema
  - El resultado **`lista-<=-qsort-qsort-<`** es demostrado directamente por el sistema

Algoritmo de ordenación rápida

- Todo elemento `e` es menor o igual que los elementos de la lista

  `(qsort (qsort-no-< e l))`

```
(defthm <=-lista-qsort-qsort-<
  (<=-lista x (qsort (qsort-no-< x l))))
```

  - La aparición consecutiva de las funciones `qsort` y `qsort-no-<` dificulta la prueba de este resultado

Algoritmo de ordenación rápida

- Una planificación para demostrar este subobjetivo consiste en usar congruencias con respecto a **`perm`**
  - **`(qsort (qsort-no-< x l))`** es una permutación de **`(qsort-no-< x l)`** (propiedad **`perm-qsort`**)
  - **`<=-lista`** es congruente con respecto a **`perm`** en su primer argumento

Algoritmo de ordenación rápida

- **<=-lista** es congruente con respecto a **perm** en su segundo argumento

```
(defthm <=-lista-perm
  (implies (perm l1 l2)
           (equal (<=-lista x l1)
                  (<=-lista x l2)))
  :rule-classes :congruence)
```

  - El resultado es demostrado directamente por el sistema
  - El resultado **<=-lista-qsort-qsort-<** es demostrado directamente por el sistema
  - El resultado **ordenada-qsort** es demostrado directamente por el sistema