Razonamiento automático
Práctica 3

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Recursión e inducción

- **Admisión de definiciones** `DEFN`

```
>(defn longitud (l)
  (if (nlistp l) 0 (add1 (longitud (cdr l)))))
...
.... inform us that the measure (COUNT L)
decreases according to the well-founded relation
LESSP in each recursive call...
...

>(defn concatena (l m)
  (if (nlistp l) m
      (cons (car l) (concatena (cdr l) m))))
...
.... inform us that the measure (COUNT L)
decreases according to the well-founded relation
LESSP in each recursive call...
...
```

- **Las funciones anteriores se admiten porque:**
  - **En cada llamada recursiva,**
  - **a uno de los argumentos** `[(CDR L)]`**,**
  - **se le puede asignar un número** `[(COUNT (CDR L))]`**,**
  - **que decrece respecto del argumento inicial** `(LESSP (COUNT (CDR L)) (COUNT L)`**, si** `(LISTP L)`
  - **y no puede existir una sucesión infinita estrictamente decreciente de números naturales.**

Esquemas de inducción

- **Al intentar mostrar algo por inducción el sistema presenta previamente el esquema de inducción:**

```
>(prove-lemma longitud-concatena (rewrite)
  (equal (longitud (concatena l m))
         (plus (longitud l) (longitud m))))
....
Perhaps we can prove it by induction......
....according to the following scheme:
      (AND (IMPLIES (NLISTP L) (p L M))
           (IMPLIES (AND (NOT (NLISTP L)) (p (CDR L) M))
                    (p L M))).
....
```

- **Un esquema de inducción indica:**
  - **los casos base del intento de prueba**
  - **los casos inductivos (donde hay hipótesis de inducción)**
  - **las hipótesis de inducción en cada caso**
- **En este ejemplo:**

```
Caso base: (NLISTP L)
Caso inductivo: (LISTP L), con h. de i. (p (CDR L) M)
```

Recursión e inducción

- **Demostración por inducción:**

```
>(prove-lemma longitud-concatena (rewrite)
  (equal (longitud (concatena l m))
         (plus (longitud l) (longitud m))))
....
     Perhaps we can prove it by induction.  Three
inductions  are suggested by terms in the conjecture.
They merge into two likely candidate inductions.
However, only one is unflawed.  We will induct
according to the following scheme:
      (AND (IMPLIES (NLISTP L) (p L M))
           (IMPLIES (AND (NOT (NLISTP L)) (p (CDR L) M))
                    (p L M))).
....
```

- **Se escoge tal esquema de inducción porque:**
  - **Las funciones involucradas en lema son** `longitud` **y** `concatena`**,**
  - **ambas funciones fueron admitidas porque se descubrió que en en las llamadas recursivas se llamaba a** `(CDR L)` **cuando** `(LISTP L)`**,**
  - **y** `(LESSP (COUNT (CDR L)) (COUNT L))`**, cuando** `(LISTP L)`**.**
  - **El caso base coincide con el case base de las definiciones recursivas.**

Tipos de datos en NQTHM.

- **Un tipo de datos en NQTHM viene definido por:**

  - Una función n-aria CONSTRUCTORA.
  - Un objeto BASE (opcional).
  - Un RECONOCEDOR de tipo.
  - n funciones SELECTORAS o ACCESORAS.
  - n restricciones de TIPO para los argumentos del constructor.
  - n valores por defecto.

- **La orden** ADD-SHELL**:**

```
(ADD-SHELL constructor base reconocedor
           ((selector1 rt1 valor-defecto1))
            ....
            (selectorN rtN valor-defectoN))
```

  - Cada rtI es una restricción de tipo, de la forma (ONE-OF rec1 ... recM) ó (NONE-OF rec1 ... recM) donde cada recI es un reconocedor de un T.D. (posiblemente el mismo que se está definiendo).
  - Cada valor-defectoI es el base de un T.D. ya definido.
  - El resto de símbolos, deben ser nuevos.
  - Si no hay elemento base, usar NIL.

- **En cada momento, cada objeto que maneje NQTHM, pertenece a uno y sólo uno de los tipos definidos.**

Definiendo tipos de datos

- **Ejemplo: árboles binarios.**

```
(add-shell arbol-b vac es-arbol
   ((raiz (none-of) zero)
    (hi (one-of es-arbol) vac)
    (hd (one-of es-arbol) vac)))
```

- **Tipos de datos predefinidos:**

```
(add-shell add1 zero numberp
   ((sub1 (one-of numberp) zero)))

(add-shell cons nil listp
   ((car (none-of) zero) (cdr (none-of) zero)))

;; Además, LITATOM y NEGATIVEP.
```

- **La función `COUNT`:**

```
Dado un objeto A, la función (COUNT A) devuelve el
número de constructores que constituyen A.

Ejemplo:
========
(COUNT 4) = (COUNT (ADD1 (ADD1 (ADD1 (ADD1 (ZERO)))))) = 4
(COUNT '(1 . 2)) =
    (COUNT (CONS (ADD1 (ZERO)) (ADD1 (ADD1 (ZERO))))) = 4
(COUNT (ARBOL-B 1 (ARBOL-B 2 (VAC) (VAC)) (VAC))) =
(COUNT (ARBOL-B (ADD1 (ZERO))
    (ARBOL-B (ADD1 (ADD1 (ZERO))) (VAC)(VAC)) (VAC))) = 5
```

Observaciones

- **Los valores por defecto permiten que las funciones constructoras y accesoras estén definidas para cualquier elemento, sea cual sea su tipo.**
- **Respecto de los accesores:**

```
Cuando un accesor actúa sobre un elemento que no es del
tipo esperado, devuelve el correspondiente valor
por defecto:
Ejemplo:
========
El tipo esperado del argumento de la función CAR es LISTP.
Si CAR actúa sobre algo que no es LISTP, devuelve el valor
por defecto definido en el ADD-SHELL (es decir, (ZERO) ó 0).
P.ej. (CAR 3) = 0
```

- **Respecto de los constructores:**

```
Cuando un constructor actúa sobre algún elemento que
no verifica la restricción de tipo, actúa como si recibiera
en su lugar el valor por defecto.
Ejemplo:
========
ADD1 debe recibir elementos que sean números (NUMBERP).
Si se aplica a un argumento que no es NUMBERP,
este argumento se transforma al valor por defecto
(0 ó (ZERO)).
P.ej., (ADD1 'hola) es (ADD1 0), es decir, 1.
```

Funciones sobre árboles

```
>(defn vaciop (a)
  (or (not (es-arbol a))
      (equal a (vac))))
...
VACIOP
;;; Función no recursiva. Se admite sin problemas.

>(defn simetrico (a)
  (if (vaciop a)
      a
      (arbol-b (raiz a) (simetrico (hd a))
                        (simetrico (hi a)))))
...
...the measure (COUNT A) decreases according to the
well-founded relation LESSP in each recursive call...
...
SIMETRICO

> (defn son-simetricos (a b)
   (if (vaciop a)
      (equal a b)
      (and (equal (raiz a) (raiz b))
           (son-simetricos (hi a) (hd b))
           (son-simetricos (hd a) (hi b)))))
...
...(COUNT A) decreases according to the well-founded
relation LESSP in each recursive call. ...
...
SON-SIMETRICOS
```

Recursión e inducción.

- **Las funciones son admitidas, ya que en cada llamada recursiva el o los argumentos son los hijos de `A` (izquierdo o derecho), y se tiene que:**

  1. `(LESSP (COUNT (HI A)) (COUNT A))`
  2. `(LESSP (COUNT (HD A)) (COUNT A))`

  siempre que `A` no es `VACIOP` (que es el "contexto" donde se producen las llamadas recursivas)

- **Demostrando el lema:**

```
>(prove-lemma simetrico-correcto (rewrite)
  (son-simetricos a (simetrico a)))
...
     We will appeal to induction. Two inductions are
suggested by terms in the conjecture.  However, they
merge into one likely candidate induction.
We will induct according to the following scheme:
      (AND (IMPLIES (VACIOP A) (p A))
           (IMPLIES (AND (NOT (VACIOP A))
                         (p (HD A))
                         (p (HI A)))
                    (p A))).
...
; Esquema de recursión inspirado tanto por
; SON-SIMETRICOS como por SIMETRICOS.
```

Esquema de inducción sugerido.

- **El esquema de inducción sugerido tanto por `simetrico` como por `es-simetrico`, es:**

  1. **Caso 1 (paso inductivo):**

     ```
     Supuesto que
       * A es arbol binario no vacío, i.e.,(NOT (VACIOP A)),
       * lo que se  quiere probar es cierto para (HI A),
       * lo que se  quiere probar es cierto para (HD A),
     hay que probar que:
       * es cierto para A.
     ```

  2. **Caso 2 (caso base):**

     ```
     Probar el resultado para A, cuando (VACIOP A)
     ```

- **Nótese que:**

  1. **Por cada "contexto" distinto en el que se produce una llamada recursiva, se tiene un caso distinto en el esquema de inducción correspondiente, además del caso base. En este caso, un caso inductivo, correspondiente a `(NOT (VACIOP A))`.**
  2. **En cada caso inductivo, se supone cierta la propiedad para tantos elementos como llamadas recursivas haya. En este caso, cierta para `(HI A)` y `(HD A)`.**
  3. **El caso base se obtiene como la negación de la conjunción de todos los casos inductivos. En este caso, `(VACIOP A)`**

Definiciones no admitidas.

- **La función `MERGE`:**

```
>(defn merge (l m)
  (if (not (listp l))
      m
      (if (not (listp m))
          l
          (if (lessp (car l) (car m))
              (cons (car l) (merge (cdr l) m))
              (cons (car m) (merge l (cdr m)))))))

ERROR:  The admissibility of this definition
has not been established.....
```

- **Razones posibles de la no admisión:**
  - **La función puede no terminar (probablemente en valores anómalos).**
  - **Aunque la función termina, el sistema no es capaz de probarlo.**
  - **Por defecto, sólo se intenta probar que decrece `COUNT` de cada argumento.**
- **El usuario puede ayudar al demostrador a que la definición sea admitida.**

Consejos para admitir definiciones.

- **Problema:**

```
En una llamada recursiva decrece el primer argumento:
       (MERGE (CDR L) M),
y en la otra decrece el segundo argumento,
       (MERGE L (CDR M)).
y ha de encontrarse una única medida numérica
que decrezca en cada llamada recursiva.
```

- **Intuición:**

```
En cada llamada recursiva la SUMA de las LONGITUDES
de los dos argumentos DECRECE.
```

- **Se lo decimos al sistema:**

```
>(defn merge (l m)
  (if (not (listp l)) m
      (if (not (listp m)) l
          (if (lessp (car l) (car m))
              (cons (car l) (merge (cdr l) m))
              (cons (car m) (merge l (cdr m))))))
  ((lessp (plus (longitud l) (longitud m)))))
....
...the measure (PLUS (LONGITUD L) (LONGITUD M))
decreases according to the well-founded relation
LESSP in each recursive call.
....
MERGE
```

Consejos para admitir definiciones.

- **Sintaxis:**

```
(DEFN nombre (x1 ... xN)
      cuerpo-de-la-funcion
   ((relacion expr)))

Donde:
 - "relación" indica la relación de decrecimiento,
   usualmente LESSP.
 - "expr" es una expresión (medida) en la que
   aparecen algunos de los argumentos de la
   funcion <nombre>



Con este consejo, para admitir la definición,
el sistema intenta probar que para llamada
recursiva se verifica
  (relacion expr' expr)

donde expr' es la expresión que resulta de
sustituir los argumentos en expr por los
argumentos empleados en la llamada recursiva.

En la prueba de esta fórmula se pueden suponer
ciertas todas las condiciones que son ciertas
en el momento en el que se produce la llamada
(contexto).
```

Ejemplo.

- **En el ejemplo anterior, el sistema prueba:**

```
Llamada recursiva 1
===================
(merge (cdr l) m)

- Contexto en el que se produce:
(listp l), (listp m), (lessp (car l) (car m))

- Relación que se prueba:
(lessp (plus (longitud (cdr l)) (longitud m))
       (plus (longitud l) (longitud m)))
QUE ES CIERTA (ya que l y m son listp)

Llamada recursiva 2
===================
(merge l (cdr m))

- Contexto en el que se produce:
(listp l), (listp m), (not (lessp (car l) (car m)))

- Relación que se prueba:
(lessp (plus (longitud l) (longitud (cdr m)))
       (plus (longitud l) (longitud m)))
QUE ES CIERTA (ya que l y m son listp)

POR TANTO LA FUNCION ES ADMITIDA
```

Otro ejemplo.

```
>(defn parte-entera-de-la-mitad (n)
  (if (or (zerop n) (equal n 1))
      0
      (add1 (parte-entera-de-la-mitad (sub1 (sub1 n))))))
...
...the measure (COUNT N) decreases according to the
well-founded relation LESSP in each recursive call.
...
PARTE-ENTERA-DE-LA-MITAD
>(prove-lemma mitad-por-dos ()
  (leq (times (parte-entera-de-la-mitad n) 2) n))
...
other.  We will induct according to the following scheme:
      (AND (IMPLIES (OR (ZEROP N) (EQUAL N 1))
                    (p N))
           (IMPLIES (AND (NOT (OR (ZEROP N) (EQUAL N 1)))
                         (p (SUB1 (SUB1 N))))
                    (p N))).
...
MITAD-POR-DOS
```

El consejo INDUCT

- **Los esquemas de inducción intentados por el demostrador son los sugeridos por las funciones que aparecen en la fórmula que se quiere demostrar**
- **A veces no se intenta el esquema adecuado para que la prueba tenga éxito.**
- **Ejemplo:**

```
(defn suma (m n k)
  (if (zerop m)
      (equal k n)
      (suma (sub1 m) (add1 n) k)))

(prove-lemma suma-es-la-suma ()
  (implies (and (numberp k) (suma m n k))
           (equal (plus m n) k)))
*** LA PRUEBA NO TIENE EXITO ****
```

- **El sistema intenta:**

```
(AND (IMPLIES (ZEROP M) (p M N K))
     (IMPLIES (AND (NOT (ZEROP M)) (p (SUB1 M) N K))
              (p M N K))).
```

El consejo INDUCT

- **Deberia intentar:**

```
(AND (IMPLIES (ZEROP M) (p M N K))
     (IMPLIES (AND (NOT (ZEROP M)) (p (SUB1 M) (ADD1 N) K))
              (p M N K))).
```

- **El consejo INDUCT permite que el usuario decida el esquema de inducción.**
- **Sintaxis:**

```
(prove-lemma NOMBRE ([rewrite])
         CUERPO
  ((induct (FN V1 ... Vn))))

donde FN es un función de n argumentos previamente
definida y las Vi son variables que aparecen en cuerpo.
```

- **El sistema intenta la prueba por inducción de CUERPO usando como esquema de inducción el sugerido por la definición de FN**

Usando INDUCT

- **Es bastante usual definir la función FN expresamente para obtener el esquema de inducción adecuado.**

```
(defn consejo-induccion (m n)
   (if (zerop m)
      t
      (consejo-induccion (sub1 m) (add1 n))))
```

- **Y usarla con el consejo INDUCT:**

```
(prove-lemma suma-es-la-suma ()
  (implies (and (numberp k) (suma m n k))
           (equal (plus m n) k))
   ((induct (consejo-induccion m n))))
```