Razonamiento automático
Práctica 2

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

El factorial.

- **Versión recursiva:**

```
(defn factorial (n)
  (if (zerop n)
      1
      (times n (factorial (sub1 n)))))

;; Ejemplo de traza:
;; (factorial 4) =
;; (times 4 (factorial 3)) =
;; (times 4 (times 3 (factorial 2))) =
;; (times 4 (times 3 (times 2 (factorial 1)))) =
;; (times 4 (times 3 (times 2 (times 1 (factorial 0))))) =
;; (times 4 (times 3 (times 2 (times 1 1)))) =
;; (times 4 (times 3 (times 2 1))) =
;; (times 4 (times 3 2)) = (times 4 6) = 24
```

- **Versión iterativa (recursiva final):**

```
(defn fact1 (n acc)
  (if (zerop n)
      acc
      (fact1 (sub1 n) (times n acc))))
(defn fact (n)
  (fact1 n 1))

;; Ejemplo de traza:
;; (fact 4) = (fact1 4 1) =
;; (fact1 3 (times 4 1)) = (fact1 3 4) =
;; (fact1 2 (times 3 4)) = (fact1 2 12) =
;; (fact1 1 (times 2 12)) = (fact1 1 24) =
;; (fact1 0 (times 1 24)) = (fact1 0 24) = 24
```

Algunas funciones aritméticas.

- **Definiciones ya existentes (primitivas):**

```
>(ppe 'zerop)
(DEFN ZEROP
      (X)
      (IF (EQUAL X 0)
          T
          (IF (NUMBERP X) F T)))
...
>(ppe 'plus)
(DEFN PLUS
      (X Y)
      (IF (ZEROP X)
          (FIX Y)
          (ADD1 (PLUS (SUB1 X) Y))))
>(ppe 'fix)
(DEFN FIX (X) (IF (NUMBERP X) X 0))
...
>(ppe 'times)
(DEFN TIMES
      (I J)
      (IF (ZEROP I)
          0
          (PLUS J (TIMES (SUB1 I) J))))
...
```

- **Otras primitivas:**

```
MAX, DIFFERENCE, QUOTIENT, REMAINDER
LESSP, GREATERP, GEQ, LEQ, COUNT
```

Primer intento.

- **Intento fallido:**

```
(prove-lemma fact-igual-que-factorial ()
  (equal (fact n) (factorial n)))

;;; PRIMER INTENTO FALLIDO (Cortar con C-c C-c)
```

- **Inspeccionar la prueba fallida.**
- **Esquema de inducción.**

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

;;; Induccion en los numeros naturales (en principio,
;;; es la induccion adecuada).
```

- **Paso "peligroso".**

```
(IMPLIES (AND (NUMBERP X)
              (EQUAL (FACT1 X 1) (FACTORIAL X)))
         (EQUAL (FACT1 X (PLUS 1 (TIMES X 1)))
                (PLUS (FACTORIAL X)
                      (TIMES X (FACTORIAL X))))).

  We now use the above equality hypothesis by substituting
  (FACT1 X 1) for .....
;;; VA A USAR IGUALDADES EN HIPOTESIS, PASO PELIGROSO.
```

Teoremas generales.

- **Problema:**

```
La expresion con FACT1 en la hipotesis, (FACT1 X 1),
no se puede usar en la conclusion, ya que lo que
aparece es (FACT1 X (PLUS 1 (TIMES X 1)))
```

- **Solución:**

```
En lugar de probar la relacion entre (FACT1 X 1) y
(FACTORIAL X), intentemos algo mas general:
pensemos en la relacion general entre (FACT1 X ACC) y
(FACTORIAL X).
```

- **Una posibilidad:**

```
(equal (fact1 n acc) (times acc (factorial n)))))
```

- **CONSEJO:**
  "En general, los resultados más generales se prueban más fácilmente"

Comprobación previa.

- **Antes de probarlo, veamos si esto basta:**

```
>(add-axiom fact1-relacion-factorial (rewrite)
   (equal (fact1 n acc)
          (times acc (factorial n))))
[ 0.0 0.0 0.0 ]
FACT1-RELACION-FACTORIAL

>(prove-lemma fact-igual-que-factorial ()
  (equal (fact n) (factorial n)))

This formula can be simplified, using the abbreviations
FACT1-RELACION-FACTORIAL and FACT, to:
      (EQUAL (TIMES 1 (FACTORIAL N))
             (FACTORIAL N)),
which simplifies, using linear arithmetic, to:
      T.
Q.E.D.

[ 0.0 0.0 0.0 ]
FACT-IGUAL-QUE-FACTORIAL
;;; Efectivamente, bastaria con probar FACT1-RELACION-FACTORIAL.
;;; Deshacemos ahora lo que hemos hecho:
>(ubt fact1-relacion-factorial)
FACT1-RELACION-FACTORIAL
```

- **El objetivo, por tanto, es probar:**

```
(prove-lemma fact1-relacion-factorial (rewrite)
   (equal (fact1 n acc) (times acc (factorial n))))
```

fact1-relacion-factorial es falso.

- **Prueba fallida de** `fact1-relacion-factorial`**:**

```
...
    This again simplifies, clearly, to the new conjecture:
...
      (EQUAL ACC (TIMES ACC 1)),
    which we will name *1.1.
...
;;; Aqui se la da un numero a la formula (*1.1) y se guarda
;;; para intentarla posteriormente por induccion.
;;; Es un paso "peligroso".
```

- **¿ Es cierto** `(EQUAL ACC (TIMES ACC 1))`**?:**

- **Funciones aritméticas: casos anómalos.**

EN GENERAL, TODAS LAS FUNCIONES ARITMETICAS PRIMITIVAS TRATAN ARGUMENTOS NO NUMERICOS COMO 0.

Por tanto, (TIMES ACC 1) es igual a ACC siempre que ACC sea un numero, es decir, (NUMBERP ACC)

Nuevo fact1-relacion-factorial.

- **Revisando** fact1-relacion-factorial

```
En el caso N=0 y ACC no numero, fact1-relacion-factorial
NO es cierto. Ejemplo:

>(r-loop)
*(fact1 0 'x)
 'X
*(times 'x (factorial 0))
 0
```

- **Nuevo** fact1-relacion-factorial

```
(prove-lemma fact1-relacion-factorial (rewrite)
          (implies (numberp acc)
                   (equal (fact1 n acc)
                          (times acc (factorial n)))))
```

- **Intento nuevamente fallido:**

```
    This again simplifies, trivially, to:
.....
          (IMPLIES (NUMBERP ACC)
                   (EQUAL ACC (TIMES ACC 1))),

    which we will name *1.1.
;;; A ESTA FORMULA SE LE DA NUMERO PARA APLICAR
;;; INDUCCION POSTERIORMENTE.
;;; El resultado es suficientemente interesante como para
;;; guardarlo como regla de reescritura.
```

Un primer lema auxiliar.

- **Formulamos el nuevo lema como de reescritura:**

```
>(prove-lemma times-1 (rewrite)
   (implies (numberp n)
            (equal (times n 1) n)))
...
[ 0.0 0.0 0.0 ]
TIMES-1

;;; SE PRUEBA!!!
```

- **Nótese que:**

```
EL DEMOSTRADOR MANEJA UN CONOCIMIENTO ELEMENTAL PREVIO DE ARITMETICA:
   ...
   which again simplifies, using linear arithmetic, to:
   ...
EN LA PRUEBA SOLO HAY UNA INDUCCION, Y LA MAYORIA DE
LOS PROCESOS APLICADOS SON DE SIMPLIFICACION:
   ...
     That finishes the proof of *1.  Q.E.D.
   ...
;; Solo se le ha asignado numero a la formula inicial
;; Por tanto, solo hay una induccion.
```

- **CONSEJO:**
  "Siempre que se pueda, buscar que las pruebas tengan una sola inducción y el resto simplificación"

Nuevo intento.

- **Otro intento fallido:**

```
(prove-lemma fact1-relacion-factorial (rewrite)
  (implies (numberp acc)
           (equal (fact1 n acc)
                  (times acc (factorial n)))))
```

- **Caso base resuelto:**

```
Case 2. (IMPLIES (AND (ZEROP N) (NUMBERP ACC))
                 (EQUAL (FACT1 N ACC)
                        (TIMES ACC (FACTORIAL N)))).
  This simplifies, applying TIMES-1, and expanding the
  functions ZEROP, EQUAL, FACT1, and FACTORIAL, to:
        T.
```

- **Problemas en el paso inductivo:**

```
(IMPLIES (AND (NUMBERP X)
              (EQUAL (FACT1 X (PLUS ACC (TIMES X ACC)))
                     (TIMES (PLUS ACC (TIMES X ACC))
                            (FACTORIAL X)))
              (NUMBERP ACC))
    (EQUAL (FACT1 X (PLUS ACC (TIMES X ACC)))
           (TIMES ACC (PLUS (FACTORIAL X)
                            (TIMES X (FACTORIAL X)))))).
  We use the above equality hypothesis by substituting:
 ....
;;; ATENCION: "uso de igualdades en las hipotesis" es
;;; un paso PELIGROSO.
```

Buscando lemas.

- **Inspeccionado la prueba fallida:**

```
El sistema debe ser capaz de darse cuenta que
(TIMES (PLUS ACC (TIMES X ACC)) (FACTORIAL X))) =
(TIMES ACC (PLUS (FACTORIAL X) (TIMES X (FACTORIAL X))))))

Al menos necesitamos reglas que expresen la
distributividad de la suma respecto al producto.
```

- **Reglas de distributividad suma-producto:**

```
(prove-lemma distributividad-times-plus-1 (rewrite)
         (equal (times (plus y z) x)
                (plus (times y x) (times z x))))

(prove-lemma distributividad-time-plus-2 (rewrite)
         (equal (times x (plus y z))
                (plus (times x y) (times x z))))
```

- **Se prueba:**

```
>(prove-lemma distributividad-times-plus-1 (rewrite)
         (equal (times (plus y z) x)
                (plus (times y x) (times z x))))
...
     That finishes the proof of *1.  Q.E.D.
...
;;; EL TIPO DE PRUEBA QUE BUSCAMOS.
```

Buscando lemas.

- **Se prueba:**

```
>(prove-lemma distributividad-time-plus-2 (rewrite)
         (equal (times x (plus y z))
                (plus (times x y) (times x z))))
...
     That finishes the proof of *1.  Q.E.D.
...
;;; EL TIPO DE PRUEBA QUE BUSCAMOS.
```

- **Nuevo intento:**

```
(prove-lemma fact1-relacion-factorial (rewrite)
  (implies (numberp acc)
           (equal (fact1 n acc)
                  (times acc (factorial n)))))
```

- **Por fin:**

```
...
     That finishes the proof of *1.1.1.1, which, consequently, also finishes
the proof of *1.1.1, which, consequently, also finishes the proof of *1.1,
which, in turn, also finishes the proof of *1.  Q.E.D.

[ 0.0 0.2 0.0 ]
FACT1-RELACION-FACTORIAL
```

Buscando lemas.

- **Problema:**

```
;; DEMASIADAS INDUCCIONES EN UNA SOLA PRUEBA
...
     That finishes the proof of *1.1.1.1, which, ....
... of *1.1.1, which, consquently.... of *1.1,
which, in turn,  .... of *1.  Q.E.D.
...
;; CUATRO PRUEBAS POR INDUCCION.
;; Busquemos *1.1 por si es resultado que merece la pena
;; tenerlo aparte como lema de reescritura.
```

- **Inspección de la prueba:**

```
...
(IMPLIES (AND (NUMBERP A)
              (NUMBERP Y)
              (NUMBERP X)
              (NUMBERP ACC))
       (EQUAL (PLUS A (TIMES (TIMES X ACC) Y))
              (PLUS A (TIMES ACC X Y)))),
  which we will finally name *1.1.
...
```

- **Asociatividad:**

(TIMES X Y Z) es una abreviatura de (TIMES X (TIMES Y Z))

(PLUS X Y Z) es una abreviatura de (PLUS X (PLUS Y Z))

Buscando lemas.

- **Un lema que sería de utilidad para** `*1.1`

```
(prove-lemma asociatividad-conmutatividad-times (rewrite)
  (equal (times (times a b) c) (times b a c)))
```

- **Deshacemos** `fact1-relacion-factorial`

```
>(ubt)
FACT1-RELACION-FACTORIAL
```

- **Probamos la asociatividad-conmutatividad:**

```
>(prove-lemma asociatividad-conmutatividad-times (rewrite)
   (equal (times (times a b) c) (times b a c)))
...
 That finishes the proof of *1.1, which, in turn, finishes the proof of *1.
Q.E.D.

[ 0.0 0.0 0.0 ]
ASOCIATIVIDAD-CONMUTATIVIDAD-TIMES
```

- **Se puede mejorar:**

```
...
          (EQUAL 0 (TIMES B 0)),
    which we will name *1.1.
...
```

Ultimo lema.

- **Deshacemos** asociatividad-conmutatividad-times

```
>(ubt)
ASOCIATIVIDAD-CONMUTATIVIDAD-TIMES
```

- **El último lema necesario:**

```
(prove-lemma times-0 (rewrite)
  (equal (times b 0) 0))
;;; Cuidado con el sentido en que se orienta!!!
```

- **El último lema necesario:**

```
...
     That finishes the proof of *1.  Q.E.D.
[ 0.0 0.0 0.0 ]
TIMES-0
```

- **Rehacemos** asociatividad-conmutatividad-times.

```
...
     That finishes the proof of *1.  Q.E.D.
[ 0.0 0.0 0.0 ]
ASOCIATIVIDAD-CONMUTATIVIDAD-TIMES

;;; EL TIPO DE PRUEBA QUE BUSCAMOS
```

Rehaciendo la prueba.

- **Rehacemos** `fact1-relacion-factorial`**.**

```
...
    That finishes the proof of *1.  Q.E.D.
[ 0.0 0.0 0.0 ]
FACT1-RELACION-FACTORIAL

;;; EL TIPO DE PRUEBA QUE BUSCAMOS
```

- **Probamos ahora el resultado que se buscaba inicialmente:**

```
(prove-lemma fact-igual-que-factorial ()
  (equal (fact n) (factorial n)))
```

- **La prueba es sólo con simplificación:**

```
This formula can be simplified, using the abbreviation FACT, to:
      (EQUAL (FACT1 N 1) (FACTORIAL N)),
which simplifies, applying the lemma FACT1-RELACION-FACTORIAL, to the goal:
      (EQUAL (TIMES 1 (FACTORIAL N)) (FACTORIAL N)).
This again simplifies, using linear arithmetic, to:
      T.
Q.E.D.

[ 0.0 0.0 0.0 ]
FACT-IGUAL-QUE-FACTORIAL
```

Fichero practica-2.events

```
(boot-strap nqthm)

(defn factorial (n)
  (if (zerop n)
      1
      (times n (factorial (sub1 n)))))

(defn fact1 (n acc)
  (if (zerop n)
      acc
      (fact1 (sub1 n) (times n acc))))

(defn fact (n)
  (fact1 n 1))

(prove-lemma times-1 (rewrite)
  (implies (numberp n)
           (equal (times n 1) n)))

(prove-lemma distributividad-times-plus-1 (rewrite)
         (equal (times (plus y z) x)
                (plus (times y x) (times z x))))

(prove-lemma distributividad-times-plus-2 (rewrite)
         (equal (times x (plus y z))
                (plus (times x y) (times x z))))
```

Fichero practica-2.events

```
(prove-lemma times-0 (rewrite)
  (equal (times b 0) 0))

(prove-lemma asociatividad-conmutatividad-times (rewrite)
  (equal (times (times a b) c)
         (times b a c)))

(prove-lemma fact1-relacion-factorial (rewrite)
  (implies (numberp acc)
           (equal (fact1 n acc)
                  (times acc (factorial n)))))

(prove-lemma fact-igual-que-factorial ()
  (equal (fact n) (factorial n)))

;;; Opcional
(make-lib "practica-2")
```

- **Usando** prove-file

```
(prove-file "practica-2")
```

- **Usando** note-lib

```
(note-lib "practica-2")
```

Conclusiones:

- **Resultados más generales se prueban mejor que los particulares.**
- **El desarrollo de una prueba es tarea del demostrador y el usuario.**
- **El usuario coopera con el demostrador suministrando los lemas adecuados.**
- **Buscamos pruebas con sólo una inducción y principalmente con simplificación.**
- **Buscamos los lemas necesarios inspeccionando en las pruebas fallidas:**
  - **Justo antes de hacer una inducción (cuando se asigna número a una fórmula).**

```
...which we will finally name *1.1.
```

  - **Justo de antes de aplicar "generalización".**

```
...which we generalize by replacing ....
```

  - **Justo antes de aplicar "uso de igualdades en hipótesis".**

```
..We now use the above equality hypothesis by substituting:...
```