Introducción al demostrador de B. & M. (NQTHM)

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Introducción

- **NQTHM: demostrador de Boyer y Moore.**
- **Demostrador de teoremas.**
- **Probar propiedades sobre programas LISP y otros lenguajes:**
  - **formalmente.**
  - **automáticamente.**
- **Métodos formales aplicados a la IS.**
- **NQTHM está programado en LISP:**
  - **mismo entorno que LISP.**
  - **LISP + funciones propias del demostrador.**
- **Similitudes y diferencias con LISP.**
- **Principales técnicas de prueba:**
  - **Inducción.**
  - **Simplificación.**
- **Sucesor: ACL2.**

Aplicaciones.

- **Matemáticas y Lógica:**
  - **Teorema de Gauss (cuadrados recíprocos).**
  - **Teorema de tautología.**
  - **Teorema de incompletitud de Gödel.**
  - **Teorema de Ramsey.**
  - **Teorema de Church-Rosser.**
- **Verificación de programas:**
  - **algoritmo de búsqueda rápida de cadenas.**
  - **compilador de expresiones.**
  - **código máquina generado por compiladores.**
  - **control de ruta de vehículos.**
- **Verificación de hardware (industria):**
  - **protocolos de comunicaciones.**
  - **microprocesadores.**

Fórmulas

- **Estilo Lisp:**

```
(equal (reverse (append a b))
       (append (reverse b) (reverse a)))

(implies (properp x)
         (equal (reverse (reverse x)) x))
```

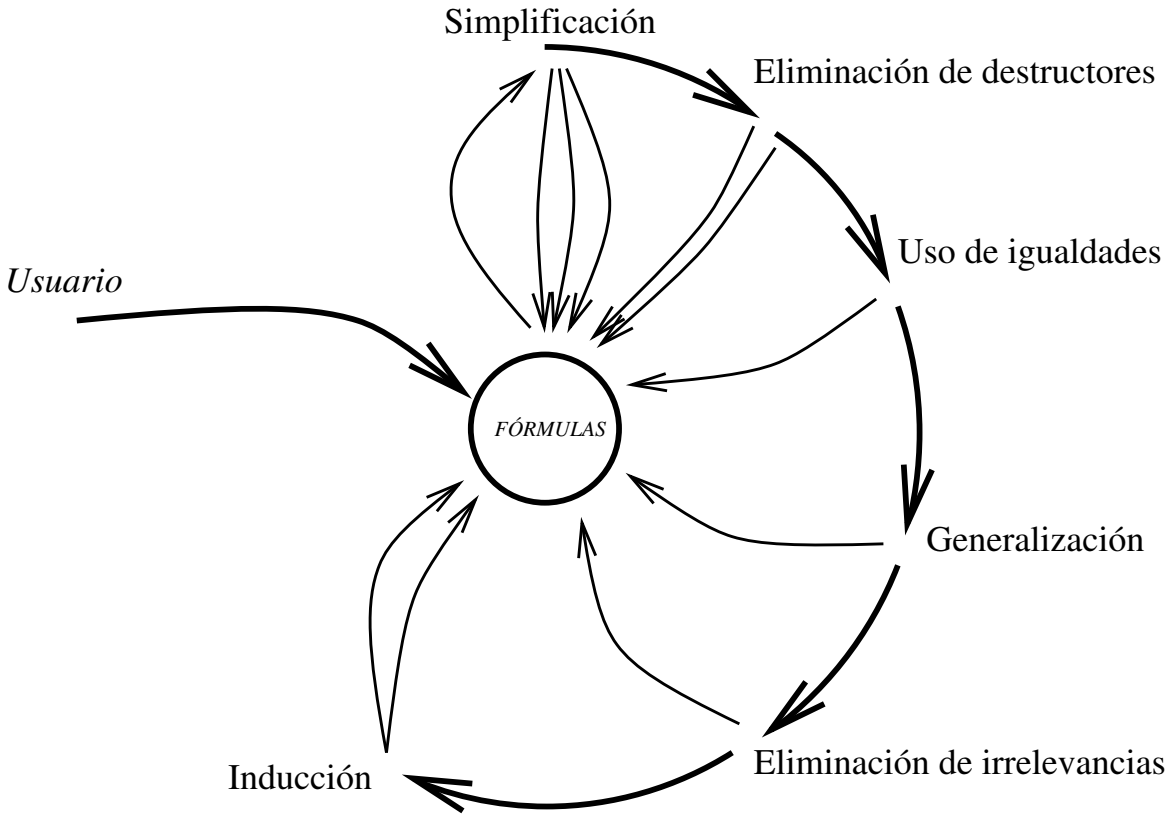
- **El comando** PROVE-LEMMA**:**

```
(prove-lemma reverse-append (rewrite)
  (equal (reverse (append a b))
         (append (reverse b) (reverse a))))

(prove-lemma reverse-reverse (rewrite)
  (implies (properp x)
           (equal (reverse (reverse x)) x)))
```

- **El comando** DEFN**:**

```
(defn reverse (x)
  (if (nlistp x)
      nil
    (append (reverse (cdr x))
            (list (car x)))))

(defn properp (x)
  (if (nlistp x)
      (equal x nil)
    (properp (cdr x))))
```

Organización del demostrador.



Organización del demostrador.

- **Seis procesos:**
  - **Reciben una fórmula y devuelven un conjunto de fórmulas.**
  - **Las fórmulas de salida implican la de entrada.**
  - **Un objetivo se divide en subobjetivos.**
  - **Sólo los dos primeros preservan la equivalencia.**
  - **El resto son procesos "peligrosos".**
  - **Las fórmulas a probar son almacenadas en una "bolsa".**
  - **La salida de cada proceso se almacena en la "bolsa".**
- **Casos particulares:**
  - **Se descubre que es un teorema (simplificando). Salida vacía.**
  - **Un proceso no es aplicable. Salida igual a la entrada.**

Organización del demostrador.

- **Secuencia de procesos:**
  - La fórmula inicial la introduce el usario (`PROVE-LEMMA`).
  - Si un proceso no es aplicable a una fórmula, pasa al siguiente.
  - Si es aplicable, se introduce su salida en la "bolsa".
  - Una fórmula sólo llega a un proceso si los anteriores no son aplicables.
  - A veces espera antes de intentar inducción.
- **Finalización:**
  - La "bolsa" queda vacía: el objetivo inicial es teorema.
  - El sistema para y la "bolsa" no está vacía: no se sabe.
  - Es posible ciclar, pero es muy raro.
- **Salida:**
  - Una fórmula se imprime sólo *después* de que se haya procesado.
  - Esto permite una explicación detallada.

Ejemplo: REVERSE-REVERSE.

```
(defn reverse (x)
  (if (nlistp x)
      nil
    (append (reverse (cdr x))
            (list (car x)))))



(defn properp (x)
  (if (nlistp x)
      (equal x nil)
    (properp (cdr x))))



(prove-lemma append-identidad-derecha (rewrite)
  (implies (properp x)
           (equal (append x nil) x)))



(prove-lemma append-es-propio (rewrite)
  (implies (properp y)
           (properp (append x y))))
```

Ejemplo: REVERSE-REVERSE.

```
(prove-lemma properp-reverse (rewrite)
  (properp (reverse x)))



(prove-lemma asoc-app (rewrite)
  (equal (append (append x y) z)
         (append x (append y z))))



(prove-lemma reverse-append (rewrite)
  (equal (reverse (append a b))
         (append (reverse b) (reverse a))))



(prove-lemma reverse-reverse (rewrite)
  (implies (properp x)
           (equal (reverse (reverse x)) x)))
```

Ejemplo de sesión.

- **Entrada del usuario.**

```
>(prove-lemma reverse-append (rewrite)
  (equal (reverse (append a b))
         (append (reverse b) (reverse a))))
```

- **Fórmulas en la "bolsa":**

```
* Fórmula 1:
  (equal (reverse (append a b))
         (append (reverse b) (reverse a)))
```

- **Procesando fórmula 1**

```
    Call the conjecture *1.

;;;; NINGUN PROCESO PREVIO A LA INDUCCION ES APLICABLE.

  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 (AND (LISTP A) (p (CDR A) B))
                    (p A B))
           (IMPLIES (NOT (LISTP A)) (p A B))).
Linear arithmetic and the lemma CDR-LESSP can be used to prove that the
measure (COUNT A) decreases according to the well-founded relation LESSP in
each induction step of the scheme.  The above induction scheme leads to two
new goals:
```

Ejemplo de sesión.

- **Fórmulas en la "bolsa":**

```
* Fórmula 1.1:
(IMPLIES (NOT (LISTP A))
         (EQUAL (REVERSE (APPEND A B))
                (APPEND (REVERSE B) (REVERSE A)))).

* Fórmula 1.2:
(IMPLIES (AND (LISTP A)
              (EQUAL (REVERSE (APPEND (CDR A) B))
                     (APPEND (REVERSE B)
                             (REVERSE (CDR A)))))
          (EQUAL (REVERSE (APPEND A B))
                 (APPEND (REVERSE B) (REVERSE A)))),
```

- **Procesando fórmula 1.2**

```
Case 2. (IMPLIES (AND (LISTP A)
                      (EQUAL (REVERSE (APPEND (CDR A) B))
                             (APPEND (REVERSE B)
                                     (REVERSE (CDR A)))))
                 (EQUAL (REVERSE (APPEND A B))
                        (APPEND (REVERSE B) (REVERSE A)))),

;;;; SE APLICA SIMPLIFICACION.

which simplifies, applying the lemmas CAR-CONS and CDR-CONS, and opening up
the definitions of APPEND and REVERSE, to:
```

Ejemplo de sesión.

- **Fórmulas en la "bolsa":**

```
* Fórmula 1.1
* Fórmula 1.2a:
(IMPLIES (AND (LISTP A)
              (EQUAL (REVERSE (APPEND (CDR A) B))
                     (APPEND (REVERSE B)
                             (REVERSE (CDR A)))))
              (EQUAL (APPEND
                          (REVERSE (APPEND (CDR A) B))
                          (LIST (CAR A)))
              (APPEND (REVERSE B)
                      (APPEND (REVERSE (CDR A))
                              (LIST (CAR A))))))).
```

- **Procesando fórmula 1.2a**

```
       (IMPLIES (AND (LISTP A)
                     (EQUAL (REVERSE (APPEND (CDR A) B))
                            (APPEND (REVERSE B)
                                    (REVERSE (CDR A)))))
                (EQUAL (APPEND (REVERSE (APPEND (CDR A) B))
                               (LIST (CAR A)))
                       (APPEND (REVERSE B)
                               (APPEND (REVERSE (CDR A))
                                       (LIST (CAR A)))))).
;;;; SIMPLIFICACION NO ES APLICABLE, PERO SI LO ES LA ELIMINACION
;;;; DE DESTRUCTORES
Appealing to the lemma CAR-CDR-ELIM, we now replace A by (CONS Z X) to
eliminate (CDR A) and (CAR A).  This generates the goal:
```

Ejemplo de sesión.

- **Fórmulas en la "bolsa":**

```
* Fórmula 1.1
* Fórmula 1.2b:
(IMPLIES (EQUAL (REVERSE (APPEND X B))
                (APPEND (REVERSE B) (REVERSE X)))
         (EQUAL (APPEND (REVERSE (APPEND X B))
                        (LIST Z))
                (APPEND (REVERSE B)
                        (APPEND (REVERSE X) (LIST Z))))).
```

- **Procesando fórmula 1.2b**

```
       (IMPLIES (EQUAL (REVERSE (APPEND X B))
                       (APPEND (REVERSE B) (REVERSE X)))
                (EQUAL (APPEND (REVERSE (APPEND X B))
                               (LIST Z))
                       (APPEND (REVERSE B)
                               (APPEND (REVERSE X) (LIST Z))))).
;;;; NO ES APLICABLE NI SIMPLIFICACION NI ELIMINACION
;;;; PERO SI ES POSIBLE EL USO DE UNA IGUALDAD EN LAS PREMISAS
  We use the above equality hypothesis by substituting:
       (APPEND (REVERSE B) (REVERSE X))
  for (REVERSE (APPEND X B)) and throwing away the equality.  This produces
  the new conjecture:
```

Ejemplo de sesión.

- **Fórmulas en la "bolsa":**
  * Fórmula 1.1
  * Fórmula 1.2c:

```
(EQUAL (APPEND (APPEND (REVERSE B) (REVERSE X))
               (LIST Z))
       (APPEND (REVERSE B)
               (APPEND (REVERSE X) (LIST Z)))),
```

- **Procesando fórmula 1.2c**

```
       (EQUAL (APPEND (APPEND (REVERSE B) (REVERSE X))
                      (LIST Z))
              (APPEND (REVERSE B)
                      (APPEND (REVERSE X) (LIST Z)))),

;;;; SIMPLIFICACION ES APLICABLE Y ADEMAS RECONOCE
;;;; QUE LA FORMULA ES UN TEOREMA.

which further simplifies, applying ASOC-APP, to:

       T.
```

- **Fórmulas en la "bolsa":**
  * Fórmula 1.1

Ejemplo de sesión.

- **Procesando fórmula 1.1**

```
Case 1. (IMPLIES (NOT (LISTP A))
                 (EQUAL (REVERSE (APPEND A B))
                        (APPEND (REVERSE B) (REVERSE A)))).

;;;; SE APLICA SIMPLIFICACION Y LA FORMULA SE RECONOCE
;;;; COMO TEOREMA

This simplifies, applying PROPERP-REVERSE and APPEND-IDENTIDAD-DERECHA, and
opening up APPEND and REVERSE, to:

        T.
```

- **Fórmulas en la "bolsa":**
  NINGUNA

- **Por tanto, la fórmula inicial es un teorema:**

```
    That finishes the proof of *1.  Q.E.D.


[ 0.0 0.4 0.1 ]
REVERSE-APPEND
```

Representación clausal.

- **Las fórmulas se convierten internamente a clausulas.**
- **Ejemplo:**

  La fórmula:

```
(IMPLIES (AND (NOT (NLISTP X))
              (EQUAL (APPEND (CDR X) NIL) (CDR X))
              (PROPERP X))
         (EQUAL (APPEND X NIL) X)).
```

  se representa internamente como la lista cuyos elementos son los literales:

```
(NLISTP X)
(NOT (EQUAL (APPEND (CDR X) NIL) (CDR X)))
(NOT (PROPERP X))
(EQUAL (APPEND X NIL) X)
```

- **Proceso transparente al usuario.**
- **Tratamiento simétrico de los literales.**
- **Papel especial de las conectivas lógicas.**

La base de datos de las reglas.

- **Se añaden reglas cada vez que:**
  - **Se definen funciones (`defn`).**
  - **Se prueban lemas (`prove-lemma`).**
  - **Se añade un nuevo tipo de datos (`add-shell`).**
  - **Se añaden axiomas (`add-axiom`).**
- **Un *evento* es el acto de añadir o quitar una regla de la base de datos.**
- **Eliminación de reglas:**
  - **Han de ser eliminadas todas las reglas de eventos posteriores a ella.**
  - **Así se mantiene la consistencia.**
- **Comandos de modificación de la base de datos:**
  - **Consulta: `ch`, `ppe`.**
  - **Eliminación: `ubt`, `undo-name`.**
- **Ficheros y librerías: `prove-file`, `note-lib`, `make-lib`.**

El papel del usuario.

- **Las reglas activas en cada momento son usadas por el demostrador como herramienta para realizar demostraciones.**
- **El proceso de demostración NO es interactivo.**
- **Sin embargo, el usuario puede influir en la prueba creando la base de datos de reglas disponibles:**
  - **Definiciones.**
  - **Reglas obtenidas a partir de resultados previamente demostrados.**
  - **Habilitando y desabilitando información.**
  - **Dando consejos (`HINT`).**
- **Una prueba se obtiene estudiando los fallos del sistema.**
- **Hay que evitar los procesos "peligrosos".**
- **Se consiguen pruebas en las que mayoritariamente se aplica inducción y simplificación.**