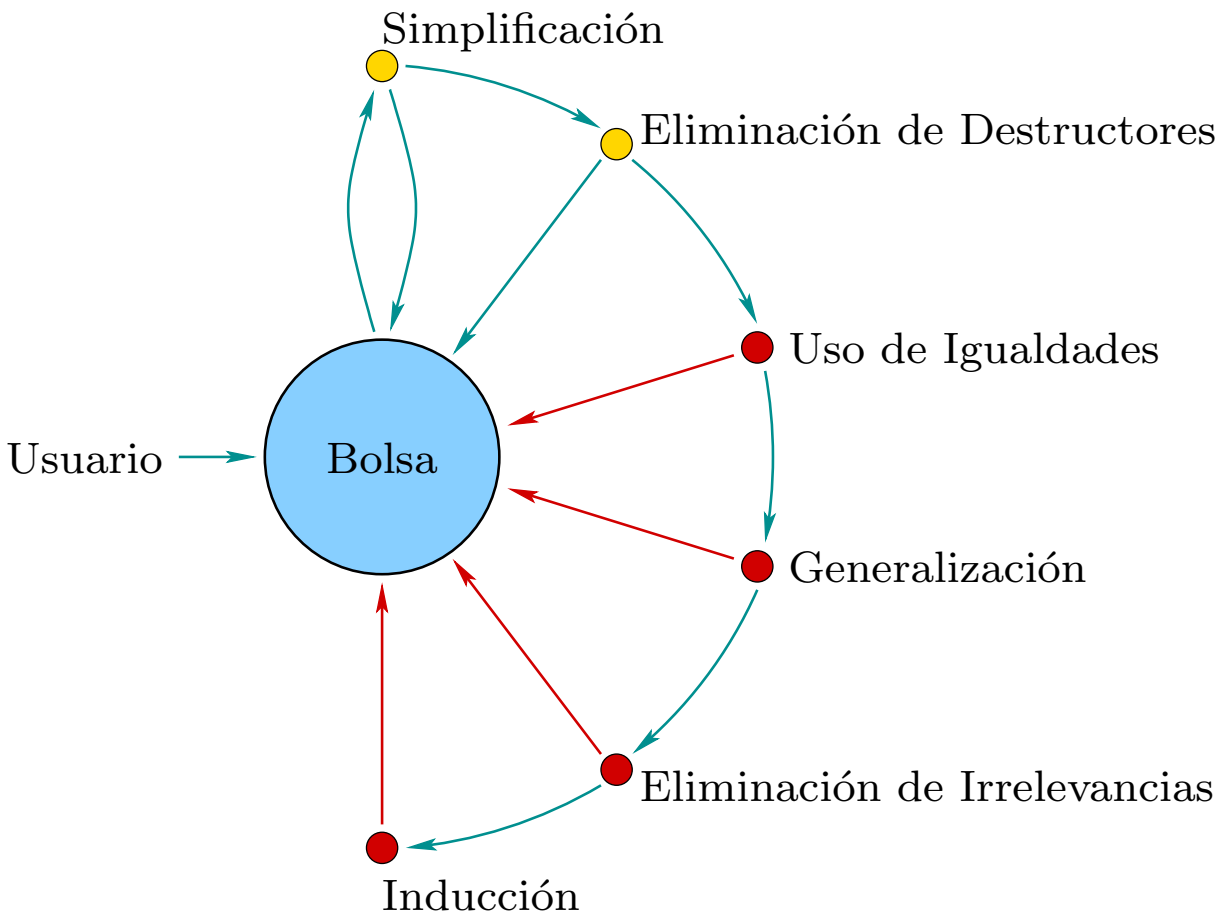
ACL2: Reescritura

Francisco J. Martín Mateos
José L. Ruiz Reina

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

Simplificación



Simplificación

- Es uno de los principales procesos de transformación que sufren las fórmulas en el demostrador
- Es un proceso "no peligroso", la fórmula resultado es equivalente a la original
- Es el único proceso que puede decidir que una fórmula es cierta
- Incluye diferentes técnicas de prueba
  - Reescritura
  - Procedimiento de decisión para el cálculo proposicional
  - Cálculo lógico con la igualdad
  - Aritmética lineal
- La simplificación trabaja a nivel de cláusulas

Representación clausal

- Las fórmulas se convierten internamente a clausulas

```
(implies (and (not (endp x))
              (equal (concatena (cdr x) nil) (cdr x))
              (lista-propia x))
         (equal (concatena x nil) x))
```

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

- `(endp x)`
- `(not (equal (concatena (cdr x) nil) (cdr x)))`
- `(not (lista-propia x))`
- `(equal (concatena x nil) x)`

- Es un proceso transparente al usuario
- Permite un tratamiento simétrico de los literales

Simplificación: Reescritura

- Es la técnica más importante dentro del proceso de simplificación. Consiste en reemplazar los subtérminos de las fórmulas por otros términos equivalentes
- Usa un *sistema de reglas de reescritura*, formado por:
  - Lemas y definiciones primitivas del sistema
  - Definiciones de funciones
  - Lemas de reescritura demostrados previamente por el usuario

Simplificación: Reescritura con premisas

- Al reescribir un literal de una cláusula, se pueden suponer falsos el resto de literales de dicha cláusula
- Fórmula inicial: `(implies (and (not p) (not q)) r)`
  - Cláusula inicial: $\{\mathbf{p}, \mathbf{q}, \mathbf{r}\}$
  - $\mathbf{p}$ se reescribe a $\mathbf{p'}$ suponiendo $\mathbf{q}$ y $\mathbf{r}$ falsos: $\{\mathbf{p'}, \mathbf{q}, \mathbf{r}\}$
  - $\mathbf{q}$ se reescribe a $\mathbf{q'}$ suponiendo $\mathbf{p'}$ y $\mathbf{r}$ falsos: $\{\mathbf{p'}, \mathbf{q'}, \mathbf{r}\}$
  - $\mathbf{r}$ se reescribe a $\mathbf{r'}$ suponiendo $\mathbf{p'}$, $\mathbf{q'}$ falsos: $\{\mathbf{p'}, \mathbf{q'}, \mathbf{r'}\}$
  - Si alguno de $\mathbf{p'}$, $\mathbf{q'}$ ó $\mathbf{r'}$ es T, el proceso de simplificación ha probado la fórmula
  - En caso contrario, la cláusula $\{\mathbf{p'}, \mathbf{q'}, \mathbf{r'}\}$ pasa a la "bolsa" de fórmulas, de donde se retomará para intentar otros procesos sobre ella. En pantalla aparece `(implies (and (not p') (not q')) r')`
  - NOTA: Puede que $\mathbf{p} = \mathbf{p'}$, ó $\mathbf{q} = \mathbf{q'}$ ó $\mathbf{r} = \mathbf{r'}$

Simplificación: Reglas de reescritura

- Reglas de reescritura incondicionales

(EQUAL $\mathcal{L}$ $\mathcal{R}$)

  - Notación: $\mathcal{L} \Rightarrow \mathcal{R}$
  - Simplificación usando reglas de reescritura incondicionales: dada una sustitución σ, reemplaza $\sigma(\mathcal{L})$ por $\sigma(\mathcal{R})$

Simplificación: Reglas de reescritura

- Reglas de reescritura condicionales (caso general)

```
(IMPLIES (AND C1
              ...
              Cn)
         (EQUAL L R))
```

  - Notación: $\mathcal{C}_1, ..., \mathcal{C}_n \rightarrow \mathcal{L} \Rightarrow \mathcal{R}$
  - Simplificación usando reglas de reescritura condicionales: dada una sustitución σ, reemplaza $\sigma(\mathcal{L})$ por $\sigma(\mathcal{R})$; estableciendo previamente las condiciones $\sigma(\mathcal{C}_1), \ldots, \sigma(\mathcal{C}_n)$
  - El uso de reglas de reescritura condicionales genera nuevos subobjetivos
  - Las reglas de reescritura incondicionales son un caso particular de las reglas de reescritura condicionales en el que $\mathbf{n = 0}$

Ejemplos de reglas de reescritura

- Regla incondicional

```
(equal (len (concatena x y))
       (+ (len x) (len y)))
```

- Regla condicional

```
(implies (lista-propia l)
         (equal (concatena l nil) l))
```

- Regla con igualdad implícita

```
(implies (and (member x l)
              (contenida l m))
         (member x m))            ; <- (equal (member x m) t)
```

Ejemplos de reglas de reescritura

- Definición no recursiva

```
(defun concatena-rev (x y)
  (concatena x (rev y)))
```

  - Regla incondicional asociada

```
(equal (concatena-rev x y)
       (concatena x (rev y)))
```

Ejemplos de reglas de reescritura

- Definición recursiva

```
(defun lista-propia (x)
  (if (endp x)
      (equal x nil)
      (lista-propia (cdr x))))
```

  - Regla incondicional asociada

```
(equal (lista-propia x)
       (if (endp x)
           (equal x nil)
           (lista-propia (cdr x))))
```

- Se usan cuando "aportan" algo a la demostración. Por ejemplo, cuando el término de la llamada recursiva aparece entre el resto de las conjeturas

Generación de reglas

- Un mismo resultado se puede expresar de maneras distintas, dando lugar a distintas reglas
  - Propiedad: Si `A` y `B`, entonces `T1` es igual que `T2`
  - Distintas reglas

```
(defthm regla-1
  (implies (and A B) (equal T1 T2)))

(defthm regla-2
  (implies (and A B) (equal T2 T1)))

(defthm regla-3
  (implies (and A (not (equal T1 T2)))
           (not B)))

(defthm regla-4
  (implies (and B (not (equal T1 T2)))
           (not A)))
```

Generación de reglas

- Consejos para formar las reglas
  - El usuario debe decidir de qué manera orientar la regla, y qué literales forman las hipótesis y cuál la conclusión
  - Principio general: la regla debe servir para reescribir términos complicados en "términos simples" (más primitivas, formas normales, menor tamaño)

Generación de reglas

- Propiedad: Si `(consp x)` entonces `(ult (reverse x))` es igual a `(car x)`
- Posibles reglas

```
(defthm regla-adecuada
  (implies (consp x)
           (equal (ult (reverse x)) (car x))))

(defthm regla-inadecuada-1
  (implies (consp x)
           (equal (car x) (ult (reverse x)))))

(defthm regla-inadecuada-2
  (implies (not (equal (ult (reverse x)) (car x)))
           (not (consp x))))
```

Simplificación: Uso de las reglas de reescritura

- Conjetura: $\mathcal{H}_1, \ldots, \mathcal{H}_m \rightarrow \mathcal{D}$
- La simplificación se realiza tanto en la conclusión $\mathcal{D}$ como en las hipótesis $\mathcal{H}_i$
- Uso de reglas de reescritura incondicionales $\mathcal{L} \Rightarrow \mathcal{R}$:
  - Si existe una instancia de $\mathcal{L}$ en $\mathcal{D}$ se obtiene la siguiente conjetura: $\mathcal{H}_1, \ldots, \mathcal{H}_m \rightarrow \mathcal{D}[\sigma(\mathcal{R})]$
  - Si existe una instancia de $\mathcal{L}$ en $\mathcal{H}_i$ se obtiene la siguiente conjetura: $\mathcal{H}_1, \ldots, \mathcal{H}_i[\sigma(\mathcal{R})], \ldots, \mathcal{H}_m \rightarrow \mathcal{D}$
- Se pueden realizar varios pasos de reescritura con la misma o distinta regla de reescritura en una misma conjetura

Simplificación: Uso de las reglas de reescritura

- Con la regla de reescritura

```
(defthm f=g
  (equal (f x) (g x)))
```

la conjetura

```
ℋ1 : (f (+ x 1))
 𝒞 : (equal (f (+ x 2)) (+ x 2))
```

se reescribe en

```
ℋ1 : (g (+ x 1))
 𝒞 : (equal (g (+ x 2)) (+ x 2))
```

Simplificación: Uso de las reglas de reescritura

- Conjetura: $\mathcal{H}_1, \ldots, \mathcal{H}_m \rightarrow \mathcal{D}$
- La simplificación se realiza tanto en la conclusión $\mathcal{D}$ como en las hipótesis $\mathcal{H}_i$
- Uso de reglas de reescritura condicionales $\mathcal{C}_1, \ldots, \mathcal{C}_n \rightarrow \mathcal{L} \Rightarrow \mathcal{R}$:
  - Si existe una instancia de $\mathcal{L}$ en $\mathcal{D}$ se obtiene la siguiente conjetura: $\mathcal{H}_1, \ldots, \mathcal{H}_m \rightarrow \mathcal{D}[\sigma(\mathcal{R})]$ si las condiciones $\sigma(\mathcal{C}_j)$ se pueden comprobar usando reescritura en la teoría ecuacional ampliada con las hipótesis $\mathcal{H}_i$
  - Si existe una instancia de $\mathcal{L}$ en $\mathcal{H}_i$ se obtiene la siguiente conjetura: $\mathcal{H}_1, \ldots, \mathcal{H}_i[\sigma(\mathcal{R})], \ldots, \mathcal{H}_m \rightarrow \mathcal{D}$ si las condiciones $\sigma(\mathcal{C}_j)$ se pueden comprobar usando reescritura en la teoría ecuacional ampliada con el resto de las hipótesis $\mathcal{H}_j$ (excepto la i-ésima) y $\neg\mathcal{D}$

Simplificación: Uso de las reglas de reescritura

- Con la regla de reescritura

```
(defthm f=g
  (implies (h x)
           (equal (f x) (g x))))
```

la conjetura

```
H1 : (h (+ x 2))
H2 : (f (+ x 2))
 C : (equal (f (+ x 2)) (+ x 2))
```

se reescribe en

```
H1 : (h (+ x 2))
H2 : (g (+ x 2))
 C : (equal (g (+ x 2)) (+ x 2))
```

Simplificación: Uso de las reglas de reescritura

- Con la regla de reescritura

```
(defthm f=g
  (implies (h x)
           (equal (f x) (g x))))
```

la conjetura

```
H1 : (f (+ x 2))
 C : (not (h (+ x 2)))
```

se reescribe en

```
H1 : (g (+ x 2))
 C : (not (h (+ x 2)))
```

Axiomas y reglas de inferencia

- Reglas proposicionales
  - *Esquema de axioma proposicional*: $(\neg\phi \vee \phi)$
  - *Regla de Expansión*: a partir de ϕ_2 se deriva $\phi_1 \vee \phi_2$
  - *Regla de Contracción*: a partir de $\phi \vee \phi$ se deriva ϕ
  - *Regla de Asociatividad*: a partir de $\phi_1 \vee (\phi_2 \vee \phi_3)$ se deriva $(\phi_1 \vee \phi_2) \vee \phi_3$
  - *Regla de Corte*: a partir de $\phi_1 \vee \phi_2$ y de $\neg\phi_1 \vee \phi_3$ se deriva $\phi_2 \vee \phi_3$

Axiomas y reglas de inferencia

- Reglas sobre igualdad
  - *Axioma de reflexividad*

    (equal x x) $\Rightarrow$ **t**
  - *Esquema de axioma de simplificación funcional*: a partir de **xi** = **yi**, y para todo símbolo de función de aridad **n**

    (equal xi yi) $\rightarrow$

    (f x1 ... xi ... xn) $\Rightarrow$ **(f x1 ... yi ... xn)**
- *Regla de Instanciación*: a partir de ϕ se deriva $\sigma(\phi)$ para cualquier sustitución σ
- Axiomas para la aritmética

Axiomas y reglas de inferencia

- Axiomatización consistente con la especificación estándar de las funciones Common Lisp
- Comportamiento de la función predefinida `if`
  - `x` $\rightarrow$ `(if x y z)` $\Rightarrow$ `y`
  - `(not x)` $\rightarrow$ `(if x y z)` $\Rightarrow$ `z`
- Comportamiento de las funciones predefinidas `car`, `cons` y `cdr` y el tipo de dato par punteado `consp`
  - `(consp nil)` $\Rightarrow$ `nil`
  - `(consp (cons x y))` $\Rightarrow$ `t`
  - `(consp x)` $\rightarrow$ `(cons (car x) (cdr x))` $\Rightarrow$ `x`
  - `(car (cons x y))` $\Rightarrow$ `x`
  - `(cdr (cons x y))` $\Rightarrow$ `y`

Ejemplo de demostración

- Definiciones

```
(defun len (l)
  (if (consp l)
      (+ 1 (len (cdr l)))
      0))

(defun concatena (l1 l2)
  (if (consp l1)
      (cons (car l1) (concatena (cdr l1) l2))
      l2))
```

- Conjetura

```
(equal (len (concatena l1 l2))
       (+ (len l1) (len l2)))
```

Ejemplo de demostración

- Inducción sugerida por `(concatena l1 l2)`
  - Caso base: $\mathbf{q_1} \equiv$ `(consp l1)`

```
(implies (not (consp l1))
         (equal (len (concatena l1 l2))
                (+ (len l1) (len l2))))
```

  - Caso de inducción: $\sigma_{1,1} = \{$`l1/(cdr l1)`$\}$

```
(implies (and (consp l1)
              (equal (len (concatena (cdr l1) l2))
                     (+ (len (cdr l1)) (len l2))))
         (equal (len (concatena l1 l2))
                (+ (len l1) (len l2))))
```

  - Medida de inducción `m`$=$`(len l1)`

Ejemplo de demostración: Caso base

- Conjetura inicial

$\mathcal{H}_1$: `(not (consp l1))`
$\mathcal{C}$:
```
(equal (len (concatena l1 l2))
       (+ (len l1)
          (len l2)))
```

- Axiomas de definición de `concatena` y `len`

```
(concatena l1 l2) ⇒
  (if (consp l1)
      (cons (car l1) (concatena (cdr l1) l2))
      l2)

(len l1) ⇒
  (if (consp l1)
      (+ 1 (len (cdr l1)))
      0)
```

Ejemplo de demostración: Caso base

- Aplicando los axiomas de definición de `concatena` y `len`

```
H1 : (not (consp l1))
 C : (equal (len (if (consp l1)
                     (cons (car l1)
                           (concatena (cdr l1) l2))
                     l2))
            (+ (if (consp l1)
                   (+ 1 (len (cdr l1)))
                   0)
               (len l2))))
```

- Axiomas del condicional `if`

```
(not x) → (if x y z) ⇒ z
```

  - Se tiene $\mathcal{H}_1 =$`(not (consp l1))` y se puede aplicar la regla de reescritura anterior

Ejemplo de demostración: Caso base

- Aplicando los axiomas del condicional `if`

$\mathcal{H}_1$: `(not (consp l1))`
$\mathcal{C}$: `(equal (len l2) (+ 0 (len l2)))`

- Aplicando reglas básicas de la aritmética

$\mathcal{H}_1$: `(not (consp l1))`
$\mathcal{C}$: `(equal (len l2) (len l2))`

- Aplicando el axioma de reflexividad de `equal`

`(equal x x)` $\Rightarrow$ `t`

tenemos

$\mathcal{H}_1$: `(not (consp l1))`
$\mathcal{C}$: `t`

- La prueba termina con éxito pues uno de los literales de la cláusula se ha reescrito a `t`

Ejemplo de demostración: Caso de inducción

- Conjetura inicial

$\mathcal{H}_1$: `(consp l1)`
$\mathcal{H}_2$: `(equal (len (concatena (cdr l1) l2))`
`(+ (len (cdr l1)) (len l2)))`
$\mathcal{C}$: `(equal (len (concatena l1 l2))`
`(+ (len l1) (len l2)))`

- Axiomas de definición de `concatena` y `len`

```
(concatena l1 l2) ⇒
  (if (consp l1)
      (cons (car l1) (concatena (cdr l1) l2))
      l2)

(len l1) ⇒
  (if (consp l1)
      (+ 1 (len (cdr l1)))
      0)
```

Ejemplo de demostración: Caso de inducción

- Aplicando los axiomas de definición de `concatena` y `len`

$\mathcal{H}_1$: `(consp l1)`
$\mathcal{H}_2$: `(equal (len (concatena (cdr l1) l2))`
`(+ (len (cdr l1)) (len l2)))`
$\mathcal{C}$: `(equal (len (if (consp l1)`
`(cons (car l1)`
`(concatena (cdr l1) l2))`
`l2))`
`(+ (if (consp l1)`
`(+ 1 (len (cdr l1)))`
`0)`
`(len l2))))`

- Axiomas del condicional `if`

`x` $\rightarrow$ `(if x y z)` $\Rightarrow$ `y`

  - Se tiene $\mathcal{H}_1 =$ `(consp l1)` y se puede aplicar la regla de reescritura anterior

Ejemplo de demostración: Caso de inducción

- Aplicando los axiomas del condicional `if`

```
H1 : (consp l1)
H2 : (equal (len (concatena (cdr l1) l2))
            (+ (len (cdr l1)) (len l2)))
 C : (equal (len (cons (car l1)
                       (concatena (cdr l1) l2)))
            (+ (+ 1 (len (cdr l1))) (len l2)))
```

- Aplicando el axioma de definición de `len`

```
(len (cons (car l1)
           (concatena (cdr l1) l2))) ⇒
  (if (consp (cons (car l1)
                   (concatena (cdr l1) l2)))
      (+ 1 (len (cdr (cons (car l1)
                           (concatena (cdr l1) l2)))))
      0)
```

Ejemplo de demostración: Caso de inducción

- Aplicando el axioma de definición de **len**

```
H1 : (consp l1)
H2 : (equal (len (concatena (cdr l1) l2))
            (+ (len (cdr l1)) (len l2)))
 C : (equal (if (consp (cons (car l1)
                             (concatena (cdr l1) l2)))
                (+ 1 (len (cdr (cons (car l1)
                                     (concatena (cdr l1) l2)))))
                0)
            (+ (+ 1 (len (cdr l1))) (len l2)))
```

- Aplicando los axiomas de definición de **cons**

```
(consp (cons (car l1) (concatena (cdr l1) l2))) ⇒ t

(cdr (cons (car l1) (concatena (cdr l1) l2))) ⇒
  (concatena (cdr l1) l2)
```

Ejemplo de demostración: Caso de inducción

- Aplicando los axiomas de definición de **cons**

```
H1 : (consp l1)
H2 : (equal (len (concatena (cdr l1) l2))
            (+ (len (cdr l1)) (len l2)))
 C : (equal (if t
                (+ 1 (len (concatena (cdr l1) l2)))
                0)
            (+ (+ 1 (len (cdr l1))) (len l2)))
```

- Axiomas del condicional **if**

```
x → (if x y z) ⇒ y
```

  - En este caso la hipótesis de la regla de reescritura condicional se tiene trivialmente

Ejemplo de demostración: Caso de inducción

- Aplicando los axiomas del condicional `if`

$\mathcal{H}_1$: `(consp l1)`
$\mathcal{H}_2$: `(equal (len (concatena (cdr l1) l2))`
`(+ (len (cdr l1)) (len l2)))`
$\mathcal{C}$: `(equal (+ 1 (len (concatena (cdr l1) l2)))`
`(+ (+ 1 (len (cdr l1))) (len l2)))`

- Aplicando la hipótesis $\mathcal{H}_2$

Ejemplo de demostración: Caso de inducción

- Aplicamos la hipótesis $\mathcal{H}_2$

$\mathcal{H}_1$: `(consp l1)`
$\mathcal{H}_2$: `(equal (len (concatena (cdr l1) l2))`
`(+ (len (cdr l1)) (len l2)))`
$\mathcal{C}$: `(equal (+ 1 (+ (len (cdr l1)) (len l2)))`
`(+ (+ 1 (len (cdr l1))) (len l2)))`

- Aplicando la asociatividad de +

Ejemplo de demostración: Caso de inducción

- Aplicando la asociatividad de `+`

```
H1 : (consp l1)
H2 : (equal (len (concatena (cdr l1) l2))
            (+ (len (cdr l1)) (len l2)))
 C : (equal (+ 1 (+ (len (cdr l1)) (len l2)))
            (+ 1 (+ (len (cdr l1)) (len l2))))
```

- Aplicando el axioma de reflexividad de `equal`

```
(equal (+ 1 (+ (len (cdr l1)) (len l2)))
       (+ 1 (+ (len (cdr l1)) (len l2)))) ⇒ t
```

tenemos

```
H1 : (consp l1)
H2 : (equal (len (concatena (cdr l1) l2))
            (+ (len (cdr l1)) (len l2)))
 C : t
```

Congruencias

- Propiedades de la igualdad
  - *Relación de equivalencia*: *reflexiva*, *simétrica* y *transitiva*
  - *Esquema de axioma de simplificación funcional*:
    $$\left.\begin{array}{l} x_1 = y_1 \\ \quad \ldots \\ x_n = y_n \end{array}\right\} \Rightarrow f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$
- Disponer del mismo esquema de simplificación funcional para otras relaciones de equivalencia ($\simeq, \approx$)
  - *Esquema general de axioma de simplificación funcional*:
    $x_i \simeq y_i \Rightarrow f(x_1, \ldots, x_i, \ldots, x_n) \approx f(y_1, \ldots, y_i, \ldots, y_n)$
  - El sistema utiliza esta congruencia para reemplazar $f(x_1, \ldots, x_i, \ldots, x_n)$ por $f(y_1, \ldots, y_i, \ldots, y_n)$ siempre que aparezca como argumento de $\approx$ y se pueda establecer la condición $x_i \simeq y_i$

Congruencias

- Demostrar que un predicado `perm` cumple las propiedades de las relaciones de equivalencia

```
(defthm perm-reflexivo
  (perm x x))

(defthm perm-simetrico
  (implies (perm x y)
           (perm y x)))

(defthm perm-transitivo
  (implies (and (perm x y)
                (perm y z))
           (perm x z)))
```

- Establecer el predicado `perm` como relación de equivalencia

```
(defequiv perm)
```

Congruencias

- Una congruencia establece un esquema de simplificación funcional para una símbolo de función y unas relaciones de equivalencia concretas

```
(defthm len-perm
  (implies (perm l1 l2)
           (equal (len l1) (len l2)))
  :rule-classes :congruence)

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

Bibliografía

- Kaufmann, M. and Manolios, P. and Moore, J S. *Computer–Aided Reasoning: An Approach*. (Kluwer Academic Publishers, 2000)
- Kaufmann, M. and Moore, J S. ACL2 *Version 5.0*. (`http://www.cs.utexas.edu/users/moore/acl2/`)