



PROGRAMACIÓN FUNCIONAL

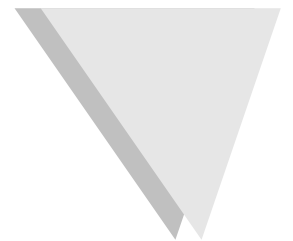
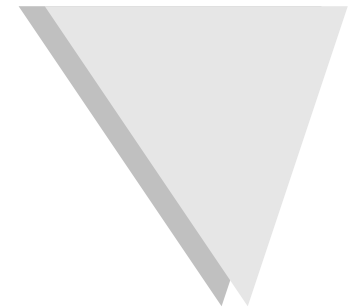
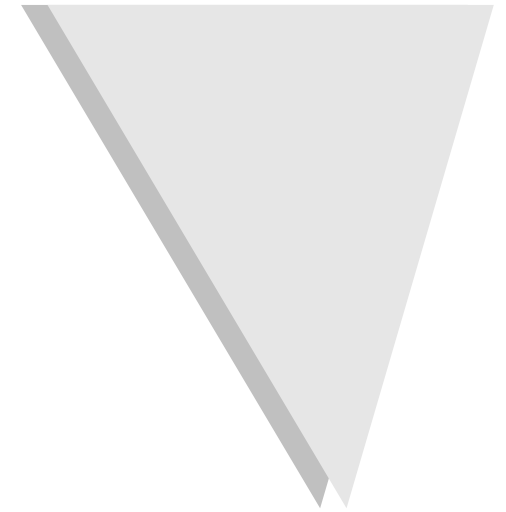
Técnicas Formales: Inducción/Recursión



Técnicas Formales

◆ Inducción/Recursión

- ◆ Definición inductiva de conjuntos
- ◆ Demostración por inducción estructural
- ◆ Definición de funciones recursivas
- ◆ Ejemplos



Otros conjuntos

- ◆ Los tipos definidos hasta ahora, ¿son suficientes para la tarea de programar?
 - ◆ por ejemplo, si estuviéramos programando un intérprete para un lenguaje de programación, ¿cómo representaríamos un programa?
- ◆ ¿Cómo definimos conjuntos con infinitos elementos?
- ◆ ¿Cómo definimos funciones recursivas que no se cuelguen?
- ◆ ¿Cómo probamos propiedades de estos conjuntos?

Inducción/Recursión

- ◆ Para solucionar los tres problemas, usaremos INDUCCIÓN
- ◆ La inducción es un mecanismo que nos permite:
 - ◆ Definir conjuntos infinitos
 - ◆ Definir funciones recursivas sobre ellos, con garantía de terminación
 - ◆ Probar propiedades sobre sus elementos

Inducción estructural

- ◆ Supongamos que quiero definir el conjunto \mathbb{N} de todas las cadenas formadas por letras **S**, y terminadas en **Z**. ¿Cómo lo hago?
- ◆ **Def:** sea \mathbb{N} el menor conjunto que satisface las siguientes condiciones
 - ◆ **Z** $\in \mathbb{N}$
 - ◆ si $e \in \mathbb{N}$, entonces **S** $e \in \mathbb{N}$

Inducción estructural

♦ Una definición inductiva de un conjunto \mathfrak{R} consiste en dar condiciones de dos tipos:

♦ reglas base ($z \in \mathfrak{R}$)

♦ que afirman que algún elemento simple x pertenece a \mathfrak{R}

♦ reglas inductivas ($y_1 \in \mathfrak{R}, \dots, y_n \in \mathfrak{R} \Rightarrow y \in \mathfrak{R}$)

♦ que afirman que un elemento compuesto y pertenece a \mathfrak{R} siempre que sus partes y_1, \dots, y_n pertenezcan a \mathfrak{R} (e y no satisface otra regla de las dadas)

y pedir que \mathfrak{R} sea el menor conjunto (en sentido de la inclusión) que satisfaga todas las reglas dadas.

Inducción estructural

- ◆ ¿Qué propiedades tiene un conjunto definido por inducción estructural?
 - ◆ Tiene infinitos elementos.
 - ◆ Todos sus elementos, o bien satisfacen una regla base, o bien satisfacen una regla inductiva.
 - ◆ Todos sus elementos son finitos.
 - ◆ El orden basado en "es parte de" es bien fundado (o sea, toda cadena descendente es finita).
(Ej.: **Z** es parte de **SZ**, **SZ** es parte de **SSZ**, etc.)

Ejemplo

♦ Sea XBE definido inductivamente como:

♦ $\Delta \in XBE$

♦ $\nabla \in XBE$

♦ si $e_1, e_2 \in XBE$, entonces $\#e_1 @e_2 \parallel \in XBE$

♦ si $e \in XBE$, entonces $\#\$e \parallel \in XBE$

♦ ¿Qué elementos tiene XBE ?

♦ ¿El elemento $\#\$ \# \nabla @ \Delta \parallel \parallel \parallel$ está en XBE ?

Ejemplo

- ♦ Sea BE definido inductivamente como:
 - ♦ $TT \in BE$
 - ♦ $FF \in BE$
 - ♦ si $e_1, e_2 \in BE$, entonces $(e_1 \wedge e_2) \in BE$
 - ♦ si $e \in BE$, entonces $(\sim e) \in BE$
- ♦ ¿Qué elementos tiene BE ? ¿Qué diferencias observa con XBE ?
 - ♦ ¿El elemento $(\sim(FF \wedge TT))$ está en BE ?

Recursión

- ◆ ¿Cómo definimos funciones sobre los elementos de *BE* o *XBE*?
- ◆ ¡Utilizando el valor de la misma función en los casos anteriores!
- ◆ ¿Y esto funciona?
- ◆ Sí, pues el orden "es parte de" es bien fundado, y por lo tanto nunca hay reducciones infinitas.

Recursión

◆ Ejemplo:

$\text{evalx} :: XBE \rightarrow \text{Int}$

$\text{evalx } \Delta = \dots$

$\text{evalx } \nabla = \dots$

$\text{evalx } \#e_1 @e_2 \parallel = \dots \text{evalx } e_1 \dots \text{evalx } e_2 \dots$

$\text{evalx } \$e \parallel = \dots \text{evalx } e \dots$

(**Atención:** esta sintaxis no es correcta, pues XBE no se puede definir así en Haskell; la misma sirve sólo a los efectos del ejemplo.)

Recursión

◆ Ejemplo:

$\text{evalx} :: XBE \rightarrow \text{Int}$

$\text{evalx } \Delta = 1$

$\text{evalx } \nabla = 0$

$\text{evalx } \#e_1 @e_2 \parallel = \text{evalx } e_1 + \text{evalx } e_2$

$\text{evalx } \$e \parallel = 1 - \text{evalx } e$

(**Atención:** esta sintaxis no es correcta, pues XBE no se puede definir así en Haskell; la misma sirve sólo a los efectos del ejemplo.)

Recursión

◆ Ejemplo:

$\text{eval} :: BE \rightarrow \text{Bool}$

$\text{eval } TT = \text{True}$

$\text{eval } FF = \text{False}$

$\text{eval } (e_1 \wedge e_2) = \text{eval } e_1 \ \&\& \ \text{eval } e_2$

$\text{eval } (\sim e) = \text{not } (\text{eval } e)$

(**Atención:** esta sintaxis no es correcta, pues *BE* no se puede definir así en Haskell; la misma sirve sólo a los efectos del ejemplo.)

Recursión

◆ Ejemplo:

$\text{nleftp} :: BE \rightarrow \text{Int} \quad \text{-- o } \text{nhash} :: XBE \rightarrow \text{Int}$

$\text{nleftp } TT = 0$

$\text{nleftp } FF = 0$

$\text{nleftp } (e_1 \wedge e_2) = 1 + \text{nleftp } e_1 + \text{nleftp } e_2$

$\text{nleftp } (\sim e) = 1 + \text{nleftp } e$

(**Atención:** esta sintaxis no es correcta, pues *BE* no se puede definir así en Haskell; la misma sirve sólo a los efectos del ejemplo.)

Funciones recursivas

- ◆ Sea S un conjunto inductivo, y T uno cualquiera. Una definición recursiva *estructural* de una función $f :: S \rightarrow T$ es una definición de la siguiente forma:
 - ◆ por cada elemento base z , el valor de $(f\ z)$ se da directamente usando valores previamente definidos
 - ◆ por cada elemento inductivo y , con partes inductivas y_1, \dots, y_n , el valor de $(f\ y)$ se da usando valores previamente definidos y los valores $(f\ y_1), \dots, (f\ y_n)$.

Principio de inducción

- ◆ ¿Cómo demostramos la siguiente propiedad de los elementos de *XBE*?
 - ◆ $P(x) \equiv x$ tiene el doble de caracteres ' \backslash ' que de ' $\#$ '.
- ◆ Δ y ∇ no tienen paréntesis, entonces, $P(\Delta)$ y $P(\nabla)$ se cumplen trivialmente
- ◆ Pero ¿cómo ver que $P(\# \$ e \backslash \backslash)$ se cumple?
 - ◆ ¡Debemos asumir que $P(e)$ se cumple!

Principio de inducción

- ◆ ¿Cómo demostramos la siguiente propiedad de los elementos de *BE*?
 - ◆ $P(x) \equiv x$ tiene tantos caracteres '(' como ')'.
 - ◆ **TT** y **FF** no tienen paréntesis, entonces, $P(\text{TT})$ y $P(\text{FF})$ se cumplen trivialmente
- ◆ Pero ¿cómo ver que $P(\sim e)$ se cumple?
 - ◆ ¡Debemos asumir que $P(e)$ se cumple!

Principio de inducción

- ◆ Sea S un conjunto inductivo, y sea P una propiedad sobre los elementos de S . *Si se cumple que:*
 - ◆ para cada elemento $z \in S$ tal que z cumple con una regla base, $P(z)$ es verdadero, y
 - ◆ para cada elemento $y \in S$ construido en una regla inductiva utilizando los elementos y_1, \dots, y_n , si $P(y_1), \dots, P(y_n)$ son verdaderos entonces $P(y)$ lo es,*entonces* $P(x)$ se cumple para todos los $x \in S$.
- $\forall x. P(x)$ se demostró por *inducción estructural* en x

Principio de inducción

- ◆ En la definición previa,
 - ◆ cada caso donde se prueba que $P(z)$ es verdadero para un elemento base z , se llama ***caso base***.
 - ◆ cada caso donde se prueba que $P(y)$ es verdadero asumiendo que $P(y_1), \dots, P(y_n)$ lo son, se llama ***caso inductivo***; además, $P(y)$ es la ***tesis inductiva***, y $P(y_1) \wedge \dots \wedge P(y_n)$ es la ***hipótesis inductiva***.
- ◆ Decimos que la propiedad “ $P(x)$ se cumple para todos los $x \in S$ ” se demostró por ***inducción estructural*** en x .

Principio de inducción

- ◆ **Ej.:** mostrar que para todo $e \in BE$, $P(e)$ se cumple, siendo $P(x) \equiv x$ tiene tantos caracteres '(' como ')'.
 - ◆ **Dem:** por inducción en la estructura de BE
 - ◆ **Casos base:** **TT** y **FF**
Trivialmente, pues ambos tienen cero de cada uno
 - ◆ **Casos inductivos:** $(e_1 \wedge e_2)$ y $(\sim e)$
Asumiendo que $P(e_1)$, $P(e_2)$ y $P(e)$ son verdaderos, entonces es fácil ver que $P((e_1 \wedge e_2))$ y $P(\sim e)$ también lo son.
 - ◆ Habiendo mostrado todos los casos, concluimos que $P(e)$ es verdadera para todo elemento de BE .

Ejemplo: LISTAS

- ◆ Dado un tipo cualquiera a , definimos inductivamente al conjunto $[a]$ con las siguientes reglas:
 - ◆ $[] :: [a]$
 - ◆ si $x :: a$ y $xs :: [a]$ entonces $x:xs :: [a]$
- ◆ ¿Qué elementos tiene $[\text{Bool}]$? ¿Y $[\text{Int}]$?
- ◆ Notación:
$$[x_1, x_2, x_3] = (x_1 : (x_2 : (x_3 : [])))$$

Ejemplo: LISTAS

- ◆ Definir por recursión una función `len` que cuente los elementos de una lista.

`len :: [a] -> Int`

-- Por inducción en la estructura de la lista `xs`

`len [] = ...`

`len (x:xs) = ... len xs ...`

Caso base

Caso inductivo

Aplicación inductiva de `len`

Ejemplo: LISTAS

- ◆ Definir por recursión una función `len` que cuente los elementos de una lista.

`len :: [a] -> Int`

-- Por inducción en la estructura de la lista `xs`

`len [] = 0`

`len (x:xs) = 1 + len xs`

Caso base

Caso inductivo

Aplicación inductiva de `len`

Observaciones

- ◆ Hay una ecuación por cada caso de la definición de `[a]`
- ◆ Los paréntesis son necesarios; si no dice `(len x):xs`
- ◆ La función `len` está definida para toda lista (siempre termina y da un valor definido).
Esta propiedad está garantizada por la construcción de `len`, y por la naturaleza inductiva de las listas.
- ◆ Se pueden demostrar propiedades de `len` por inducción estructural en la lista argumento.

Ejemplo: LISTAS

- ◆ **Propiedad:** demostrar que para toda lista xs , su longitud es mayor o igual que cero.
- ◆ **Dem.:** por inducción en la estructura de xs
 - ◆ Caso base: $xs = []$ — — $P([])$
 $\text{len } [] = 0$ por def. de len , por lo tanto, $\text{len } [] \geq 0$
 - ◆ Caso inductivo: $xs = x:xs'$ — — $P(xs') \Rightarrow P(x:xs')$
HI) Asumimos $\text{len } xs' \geq 0$, (hipótesis inductiva).

$$\begin{aligned}\text{len } (x:xs') &= && \text{(por def. de len)} \\ 1 + \text{len } xs' &\geq && \text{(por HI y aritmética)} \\ 1 &\geq 0\end{aligned}$$

Conclusiones

◆ Este tema cubre:

- ◆ cómo definir conjuntos infinitos con características útiles (conjuntos inductivos).
- ◆ cómo demostrar propiedades sobre los elementos de los conjuntos inductivos.
- ◆ cómo definir funciones recursivas sobre conjuntos inductivos, garantizando su terminación en todos los casos.