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 estuvieramos 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 $\aleph$ de todas las cadenas formadas por letras **S**, y terminadas en **Z**. ¿Cómo lo hago?
- **Def**: sea $\aleph$ el menor conjunto que satisface las siguientes condiciones
  - **Z** $\in \aleph$
  - si $e \in \aleph$, entonces **S**$e \in \aleph$

Inducción estructural

- Una definición inductiva de un conjunto $\Re$ consiste en dar condiciones de dos tipos:
  - reglas base ($z \in \Re$)
    - que afirman que algún elemento simple x pertenece a $\Re$
  - reglas inductivas ($y_1 \in \Re, \ldots, y_n \in \Re \Rightarrow y \in \Re$)
    - que afirman que un elemento compuesto y pertenece a $\Re$ siempre que sus partes $y_1, \ldots, y_n$ pertenezcan a $\Re$ (e y no satisface otra regla de las dadas)

  y pedir que $\Re$ 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**** $\in$ ***XBE***
  - si $e \in$ ***XBE***, entonces **#$**$e$**** $\in$ ***XBE***
- ¿Qué elementos tiene ***XBE***?
  - ¿El elemento **#$#∇@Δ** 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**^**e_2**)** $\in$ ***BE***
  - si $e \in$ ***BE***, entonces **(~**e**)** $\in$ ***BE***
- ¿Qué elementos tiene ***BE***? ¿Qué diferencias observa con ***XBE***?
    - ¿El elemento **(~(FF^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:

  evalx :: ***XBE*** -> Int
  evalx **Δ** = ...
  evalx **∇** = ...
  evalx **#**e_1**@**e_2**** = ... evalx e_1 ... evalx e_2 ...
  evalx **#$**$e$**** = ... 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:

  evalx :: ***XBE*** -> Int

  evalx Δ = 1

  evalx ∇ = 0

  evalx #e_1@e_2\\ = evalx e_1 + evalx e_2

  evalx #$$e$\\ = 1 - 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:

  eval :: ***BE*** -> Bool
  eval **TT** = True
  eval **FF** = False
  eval **(**e_1**^**e_2**)** = eval e_1 && eval e_2
  eval **(~**e**)** = not (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:

  nleftp :: ***BE*** -> Int -- o nhash :: ***XBE*** -> Int
  nleftp **TT** = 0
  nleftp **FF** = 0
  nleftp **(**e_1**^**e_2**)** = 1 + nleftp e_1 + nleftp e_2
  nleftp **(~**e**)** = 1 + 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 -> 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 y1, ..., yn, el valor de (f y) se da usando valores previamente definidos ***y*** los valores (f y1), ..., (f yn).

Principio de inducción

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

Principio de inducción

- ¿Cómo demostramos la siguiente propiedad de los elementos de ***BE***?
  - P(x) ≡ x tiene tantos caracteres '**(**' como '**)**'.
- **TT** y **FF** no tienen paréntesis, entonces, P(**TT**) y P(**FF**) se cumplen trivialmente
- Pero ¿cómo ver que P(**(~**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 construído en una regla inductiva utilizando los elementos y_1, ..., y_n, si P(y_1), ..., 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), ..., P(y_n) lo son, se llama ***caso inductivo***; además, P(y) es la ***tesis inductiva***, y P(y_1)$\wedge$...$\wedge$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**^**e_2**)** y **(~**e**)**
    Asumiendo que P(e_1), P(e_2) y P(e) son verdaderos, entonces es fácil ver que P(**(**e_1**^**e_2**)**) y P(**(~**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 [Bool]? ¿Y [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([])

    len [ ] = 0 por def. de len, por lo tanto, len [ ] ≥ 0
  - Caso inductivo: xs = x:xs' – – P(xs') ⇒ P(x:xs')

    HI) Asumimos len xs' ≥ 0 , (hipótesis inductiva).

    len (x:xs') = (por def. de len)

    1 + len xs' ≥ (por HI y aritmética)

    1 ≥ 0

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.