Inducción sobre Árboles

Á. Tasistro

La inducción, así como la recursión, tiene su generalización a los variados tipos de *árboles*. Consideremos, para comenzar, árboles binarios:

`data BinTree a = Empty | Node a (BinTree a) (BinTree a).`

¿Cómo funciona en este caso la idea de "arranque y propagación"? Bueno, debemos asegurarnos de que, para empezar, el árbol vacío `Empty` tenga la propiedad requerida. Y, luego, el constructor `Node` debe preservar (propagar) la propiedad en cuestión. Es decir, si `Node` utiliza árboles que *ya* tuvieran la propiedad, el nuevo árbol resultante de la construcción debe continuar teniendo la propiedad. De ese modo, nos aseguraremos de que *todos los árboles de tipo* `BinTree a` cumplirán la propiedad considerada. Esto nos conduce a la siguiente formulación:

<u>*Método de Demostración por Inducción en árboles binarios del tipo `BinTree a`*:</u>
<u>Sea $\mathcal{P}$ una propiedad de árboles binarios del tipo `BinTree a`. Si demostramos:</u>

1. <u>Caso Base</u>. *Tesis:* $\mathcal{P}$ `Empty`, y
2. <u>Paso Inductivo</u>. *Hipótesis:* Sean *izq* y *der* árboles del tipo `BinTree a` que cumplen la propiedad $\mathcal{P}$, es decir, tales que $\mathcal{P}$ *izq* y $\mathcal{P}$ *der*. Sea x de tipo `a`.

   Tesis: $\mathcal{P}$ (`Node` x *izq der*),

entonces podemos concluir ($\forall t \in$ `BinTree a`) $\mathcal{P}\,t$.

Nuevamente, como en los casos de naturales y listas, lo precedente es el *enunciado* del método o principio de inducción correspondiente a este tipo de árboles. Ahora podemos proceder a *aplicarlo* para demostrar propiedades de programas. Consideremos las funciones que computan la cantidad de nodos y la altura de árboles dados:

```
cant_nodos :: BinTree a -> Integer
cant_nodos Empty = 0
cant_nodos (Node x izq der) = 1 + cant_nodos izq + cant_nodos der.

altura :: BinTree a -> Integer
altura Empty = 0
altura (Node x izq der) = 1 + max (altura izq) (altura der).
```

En `altura` hemos utilizado `max` que computa el mayor de dos enteros dados. Ahora podemos probar:

($\forall t \in$ `BinTree a`) `cant_nodos` $t \geq$ `altura` t.

La propiedad a considerar es ahora:

$\mathcal{P}\,t \equiv$ `cant_nodos` $t \geq$ `altura` t,

y procedemos como de costumbre *enunciando* primeramente los casos a considerar:

<u>Caso Base</u>. *Tesis:* `cant_nodos Empty` $\geq$ `altura Empty`.

<u>Paso Inductivo</u>. *Hipótesis:* Sean *izq* y *der* árboles del tipo `BinTree a` tales que `cant_nodos` *izq* $\geq$ `altura` *izq* y `cant_nodos` *der* $\geq$ `altura` *der*. Sea x de tipo `a`.

Tesis: `cant_nodos` (`Node` x *izq der*) $\geq$ `altura` (`Node` x *izq der*).

Las demostraciones son como sigue:

<u>Caso Base</u>. *Tesis:* `cant_nodos Empty` $\geq$ `altura Empty`.

Demostración:

`cant_nodos Empty`

= (Código de `cant_nodos`)

0

$\geq$ (Reflexividad de $\geq$)
  0
$=$ (Código de `altura`)
  `altura Empty`
$\square$

Paso Inductivo. *Hipótesis:* Sean *izq* y *der* árboles del tipo `BinTree a` tales que `cant_nodos` *izq* $\geq$ `altura` *izq* y `cant_nodos` *der* $\geq$ `altura` *der*. Sea x de tipo `a`.
Tesis: `cant_nodos` (`Node` x *izq der*) $\geq$ `altura` (`Node` x *izq der*).
Demostración:
  `cant_nodos` (`Node` x *izq der*)
$=$ (Código de `cant_nodos`)
  $1 +$ `cant_nodos` *izq* $+$ `cant_nodos` *der*
$\geq$ (Dado que, por hipótesis, `cant_nodos` *izq* $\geq$ `altura` *izq* y `cant_nodos` *der* $\geq$ `altura` *der*)
  $1 +$ `altura` *izq* $+$ `altura` *der*
$\geq$ (Dado que la suma de dos naturales es mayor o igual que cualquiera de ellos, en particular, mayor o igual que el máximo)
  $1 +$ `max` (`altura` *izq*) (`altura` *der*)
$=$ (Código de `altura`)
  `altura` (`Node` x *izq der*)
$\square$

?1.

1. Programar `cant_vacios` que computa la cantidad de árboles vacíos contenidos en un árbol dado de tipo `BinTree a`.
2. Demostrar que $(\forall t \in$ `BinTree a`$)$ `cant_vacios` $t = 1 +$ `cant_nodos` t.

Para terminar, revisitemos los árboles de fórmulas aritméticas:

```
data  ExprArit  =  K Integer
                |  N ExprArit
                |  (:+) ExprArit ExprArit
                |  (:*) ExprArit ExprArit.
```

Nos interesa, como en todos los casos precedentes, primeramente *enunciar* el método de inducción correspondiente a este tipo de datos. Observamos para ello que la definición del tipo contiene:

1. Un caso base, correspondiente al constructor `K` que "envuelve" un entero convirtiéndolo en una expresión.
2. *Tres* casos recursivos, correspondiendo a los otros constructores. El constructor `N` extiende una sola expresión (con un signo o negación aritmética) en tanto los otros dos constructores representan los operadores binarios de suma y producto y, por lo tanto, arman una expresión a partir de *dos* árboles de fórmula dados.

¿Cuál es entonces el principio de inducción correspondiente al tipo `ExprArit`? Bueno, si queremos demostrar que una propiedad $\mathcal{P}$ vale *para todo* árbol de fórmula de este tipo, es suficiente garantizar que se cumplen "arranque y propagación", es decir:

1. Que cualquiera sea el entero x dado, `K` x cumple $\mathcal{P}$ (caso base o "arranque").
2. Que el constructor `N` preserva o propaga la propiedad, es decir, que siempre que se aplique a un árbol de fórmula que *ya* tuviera la propiedad, el árbol resultante de esa aplicación *también* cumplirá la propiedad.

3. Y similarmente para los dos constructores binarios —en estos casos, teniendo en cuenta que es suficiente que la propiedad se preserve cuando *los dos* árboles de fórmula combinados por cada constructor la vengan trayendo.

Estas consideraciones nos conducen al siguiente enunciado:

Método de Demostración por Inducción en árboles de fórmulas del tipo `ExprArit`:
Sea $\mathcal{P}$ una propiedad de árboles de fórmulas del tipo `ExprArit`. Si demostramos:

1. Caso Base. *Hipótesis:* Sea x de tipo `Integer`.

   Tesis: $\mathcal{P}\,(\texttt{K}\,x)$,

2. Paso Inductivo 1 (correspondiente a la negación aritmética).

   Hipótesis: Sea a un árbol de fórmula del tipo `ExprArit` que cumple la propiedad $\mathcal{P}$, es decir, tal que $\mathcal{P}\,a$.

   Tesis: $\mathcal{P}\,(\texttt{N}\ a)$,

3. Paso Inductivo 2 (correspondiente al operador de suma).

   Sean a y b árboles de fórmula del tipo `ExprArit` tales que $\mathcal{P}\,a$ y $\mathcal{P}\,b$.

   Tesis: $\mathcal{P}\,(a\ \texttt{:+}\ b)$, y

4. Paso Inductivo 3 (correspondiente al operador de producto).

   Sean a y b árboles de fórmula del tipo `ExprArit` tales que $\mathcal{P}\,a$ y $\mathcal{P}\,b$.

   Tesis: $\mathcal{P}\,(a\ \texttt{:*}\ b)$,

entonces podemos concluir $(\forall a \in \texttt{ExprArit})\ \mathcal{P}\,a$.

Las aplicaciones van ahora como ejercicios:

?2.

1. Programar `espejo` que computa la imagen en espejo de un árbol de fórmula del tipo `ExprArit` dado.

2. Programar `eval` que computa el entero denotado por un árbol de fórmula del tipo `ExprArit` dado.

3. Programar `set`, que reemplaza todos los enteros en un árbol de fórmula del tipo `ExprArit`, por un número entero dado, dejando el resto de la fórmula sin cambiar.

4. Demostrar que:

   (a) $(\forall a \in \texttt{ExprArit})\ \texttt{espejo(espejo}\ a) = a$.

   (b) $(\forall a \in \texttt{ExprArit})\ \texttt{eval(espejo}\ a) = \texttt{eval}\ a$.

   (c) $(\forall a \in \texttt{ExprArit})\ \texttt{eval(set}\ a\ 0) = 0$.

Por último, el siguiente ejercicio plantea formular el principio de inducción correspondiente a un tipo de datos a diseñar:

?3.

1. Introducir un `data` (tipo inductivo) para las fórmulas de Lógica Proposicional formadas a partir de letras (strings) por medio de la *negación*, *conjunción* y *disyunción*.

2. Formular el *método de demostración por inducción* correspondiente al tipo precedente.