Inducción sobre Listas
Á. Tasistro
Primavera de 2016

El método de inducción puede generalizarse a las listas de cualquier tipo de elementos. La clave está en observar que ellas pueden generarse de una manera similar a la empleada para los naturales, es decir, toda lista de elementos de tipo `a` es:

- O bien la lista vacía `[]`,
- o bien una lista de la forma $x : xs$, donde x es de tipo `a` y xs es una lista de elementos de tipo `a`.

(Como ya sabemos `(:)` es la función *constructora* de listas, llamada "cons".)

En consecuencia, si ahora se tiene una *propiedad de listas*, digamos $\mathcal{P}$, podemos enunciar un método de inducción que sea suficiente para demostrar que *toda lista* tiene la propiedad $\mathcal{P}$. Para ello generalizamos la idea de "arranque y propagación"; o sea:

- si la lista inicial (la vacía) tiene la propiedad $\mathcal{P}$, y
- si toda vez que se tiene una lista que *ya cumple* la propiedad y se la prolonga mediante el constructor `(:)` se obtiene como resultado una lista que *también* tiene la propiedad,

entonces necesariamente *toda lista tendrá la propiedad.*

La idea de "arranque y propagación correctos" o, en otros términos, de "inicio y herencia" es la idea común de la recursión y de la inducción, y se aplica en cualquier tipo de datos que pueda generarse a partir de "semillas" iniciales (casos base) por medio de operaciones constructoras que permiten agrandar las estructuras. Como veremos en la sección siguiente, estos tipos de datos así generados (llamados *tipos inductivos*) son, en general, tipos de árboles.

Volviendo ahora a las listas, podemos *enunciar* su principio de inducción:

Sea $\mathcal{P}$ una propiedad de listas. Si demostramos:

1. <u>Caso Base</u>. *Tesis:* $\mathcal{P}$ `[]`, y
2. <u>Paso Inductivo</u>. *Hipótesis:* Sea $xs \in$ `[a]` tal que $\mathcal{P}\,xs$. Sea $x \in$ `a`.
   Tesis: $\mathcal{P}\,(x : xs)$,

entonces podemos concluir $(\forall xs \in$ `[a]`$)\,\mathcal{P}\,xs$.

En efecto, el caso base establece el arranque correcto: la lista inicial (vacía) tiene la propiedad deseada. Y, por el otro lado, el paso inductivo asegura que todo alargamiento de una lista que ya tenía la propiedad preserva o transmite a ésta. Luego, todas las listas posibles tendrán la propiedad. Veamos ahora *aplicaciones* del principio. Para ello, repasemos algunas funciones conocidas. La primera es la que calcula el largo de una lista dada:

```
length ::  [a] -> Integer
length [] = 0
length (x : xs) = 1 + length xs.
```

La siguiente es la concatenación de dos listas:

```
(++) ::  [a] -> [a] -> [a]
[] ++ ys = ys
(x : xs) ++ ys = x : (xs ++ ys).
```

Ahora podemos probar que el largo de la concatenación de dos listas no es otra cosa que la suma de los largos de esas listas. O sea:

Proposición. $(\forall xs \in$ `[a]`$)(\forall ys \in$ `[a]`$)$ `length`$(xs +\!\!+\, ys) =$ `length` $xs\, +$ `length` ys.

Como lo hicimos en la sección precedente, el primer paso es observar que la forma de la proposición es la correcta, es decir:
$(\forall xs \in \texttt{[a]})\ \mathcal{P}\, xs$.
En otras palabras, se trata de demostrar una propiedad $\mathcal{P}$ *para toda lista*. En este caso, la propiedad es:
$\mathcal{P}\, xs \equiv (\forall ys \in \texttt{[a]})\ \texttt{length}(xs \mathbin{\texttt{++}} ys) = \texttt{length}\, xs + \texttt{length}\, ys$.
Intentando la inducción en xs, procedemos primero a enunciar los "teoremas" componentes, es decir, el caso base y el paso inductivo. Para ello efectuamos las sustituciones mecánicas correspondientes, notando que la variable a sustituir es xs:

Caso Base: *Tesis:* $(\forall ys \in \texttt{[a]})\ \texttt{length}(\texttt{[]} \mathbin{\texttt{++}} ys) = \texttt{length}\ \texttt{[]} + \texttt{length}\, ys$.

Paso Inductivo: *Hipótesis:* Sea $xs \in \texttt{[a]}$ tal que $(\forall ys \in \texttt{[a]})\ \texttt{length}(xs \mathbin{\texttt{++}} ys) = \texttt{length}\, xs + \texttt{length}\, ys$. Sea $x \in \texttt{a}$.
Tesis: $(\forall ys \in \texttt{[a]})\ \texttt{length}((x : xs) \mathbin{\texttt{++}} ys) = \texttt{length}\,(x : xs) + \texttt{length}\, ys$.

Las demostraciones pueden hacerse como sigue:

Caso Base: *Tesis:* $(\forall ys \in \texttt{[a]})\ \texttt{length}(\texttt{[]} \mathbin{\texttt{++}} ys) = \texttt{length}\ \texttt{[]} + \texttt{length}\, ys$.
Demostración: Usando la táctica de introducción del $\forall$ consideramos $ys \in \texttt{[a]}$ arbitraria y pasamos a demostrar $\texttt{length}(\texttt{[]} \mathbin{\texttt{++}} ys) = \texttt{length}\ \texttt{[]} + \texttt{length}\, ys$. Calculamos cada miembro de la igualdad por su lado, intentando llegar a una misma expresión. Comenzando por el lado izquierdo:
$\texttt{length}(\texttt{[]} \mathbin{\texttt{++}} ys)$
$=$ (Código de ++ en caso base)
$\texttt{length}\, ys$.
Ahora por el lado derecho:
$\texttt{length}\ \texttt{[]} + \texttt{length}\, ys$
$=$ (Código de length en caso base)
$0 + \texttt{length}\, ys$
$=$ (Aritmética)
$\texttt{length}\, ys$
□

Paso Inductivo: *Hipótesis:* Sea $xs \in \texttt{[a]}$ tal que $(\forall ys \in \texttt{[a]})\ \texttt{length}(xs \mathbin{\texttt{++}} ys) = \texttt{length}\, xs + \texttt{length}\, ys$. Sea $x \in \texttt{a}$.
Tesis: $(\forall ys \in \texttt{[a]})\ \texttt{length}((x : xs) \mathbin{\texttt{++}} ys) = \texttt{length}\,(x : xs) + \texttt{length}\, ys$.
Demostración: Usando la táctica de introducción del $\forall$ consideramos $ys \in \texttt{[a]}$ arbitraria y pasamos a demostrar $\texttt{length}((x : xs) \mathbin{\texttt{++}} ys) = \texttt{length}\,(x : xs) + \texttt{length}\, ys$. Nuevamente calcularemos cada miembro de la igualdad por su lado, para llegar a una expresión que los iguale. Empezamos por el lado izquierdo:
$\texttt{length}((x : xs) \mathbin{\texttt{++}} ys)$
$=$ (Código de ++, caso recursivo)
$\texttt{length}(x : (xs \mathbin{\texttt{++}} ys))$
$=$ (Código de length, caso recursivo)
$1 + \texttt{length}(xs \mathbin{\texttt{++}} ys)$
$=$ (Hipótesis de inducción, $\texttt{length}(xs \mathbin{\texttt{++}} ys) = \texttt{length}\, xs + \texttt{length}\, ys$)
$1 + \texttt{length}\, xs + \texttt{length}\, ys$.
Entretanto, por el lado derecho:
$\texttt{length}\,(x : xs) + \texttt{length}\, ys$
$=$ (Código de length, caso recursivo)
$1 + \texttt{length}\, xs + \texttt{length}\, ys$
□

Consideremos ahora esta otra clásica función:

```
filter ::  (a -> Bool) -> [a] -> [a]
filter  p []        = []
filter  p (x : xs)
        | p x       = x : filter p xs
        | not(p x)  = filter p xs
```

Podemos probar, para cualquier predicado p:

Proposición. $(\forall xs \in \texttt{[a]})\ \texttt{length}(\texttt{filter}\ p\ xs) \leq \texttt{length}\, xs$.

Es decir que la propiedad a considerar ahora es:

$\mathcal{P}\, xs \equiv \texttt{length}(\texttt{filter}\ p\ xs) \leq \texttt{length}\, xs$,

y procediendo a formular los casos de la inducción aplicando las sustituciones mecánicas se tiene:

Caso Base: *Tesis:* $\texttt{length}(\texttt{filter}\ p\ \texttt{[]}) \leq \texttt{length []}$.

Paso Inductivo: *Hipótesis:* Sea $xs \in \texttt{[a]}$ tal que $\texttt{length}(\texttt{filter}\ p\ xs) \leq \texttt{length}\, xs$. Sea $x \in \texttt{a}$.
Tesis: $\texttt{length}(\texttt{filter}\ p\ (x\!:\!xs)) \leq \texttt{length}\,(x\!:\!xs)$.

Las demostraciones se dan a continuación:

Caso Base: *Tesis:* $\texttt{length}(\texttt{filter}\ p\ \texttt{[]}) \leq \texttt{length []}$.
Demostración:
$\texttt{length}(\texttt{filter}\ p\ \texttt{[]})$
$=$ (Código de `filter`, caso base)
$\texttt{length []}$
$\leq$ (Reflexividad de $\leq$)
$\texttt{length []}$
$\square$

Paso Inductivo: *Hipótesis:* Sea $xs \in \texttt{[a]}$ tal que $\texttt{length}(\texttt{filter}\ p\ xs) \leq \texttt{length}\, xs$. Sea $x \in \texttt{a}$.
Tesis: $\texttt{length}(\texttt{filter}\ p\ (x\!:\!xs)) \leq \texttt{length}\,(x\!:\!xs)$.
Demostración: Esta vez comenzaremos calculando el lado derecho de la *in*ecuación:
$\texttt{length}\,(x\!:\!xs)$
$=$ (Código de `length`, caso recursivo)
$1 + \texttt{length}\, xs$.

Ahora tomaremos el lado izquierdo procurando llegar a una expresión que sea menor o igual que la recién alcanzada. Pero calcular el lado izquierdo requiere calcular $\texttt{filter}\ p\ (x\!:\!xs)$, y éste está definido por casos, según valga o no $p\,x$. Como regla general, la estructura de la demostración debe seguir la estructura del código y, por lo tanto, se divide en dos casos:

1. *Caso* $p\,x$:

   $\texttt{length}(\texttt{filter}\ p\ (x\!:\!xs))$

   $=$ (Código de `filter`, teniendo en cuenta que vale $p\,x$)

   $\texttt{length}(x\!:\!\texttt{filter}\ p\ xs)$

   $=$ (Código de `length`, caso recursivo)

   $1 + \texttt{length}\,(\texttt{filter}\ p\ xs)$

   $\leq$ (Dado que, por hipótesis, $\texttt{length}\,(\texttt{filter}\ p\ xs) \leq \texttt{length}\, xs$)

   $1 + \texttt{length}\, xs$,

   que es donde deseábamos arribar.

2. *Caso* $\neg(p\,x)$:

   `length`(`filter` p $(x\!:\!xs)$)

   $=$ (Código de `filter`, teniendo en cuenta que *no* vale $p\,x$)

   `length`(`filter` p xs)

   $\leq$ (Hipótesis)

   `length` xs

   $\leq$ (De hecho $<$, estrictamente)

   $1 +$ `length` xs

□

?1. Demostrar:

1. $(\forall xs \in$ `[a]`$)$ xs `++` `[]` $= xs$.
2. (Asociatividad de `++`) $(\forall xs \in$ `[a]`$)(\forall ys \in$ `[a]`$)(\forall zs \in$ `[a]`$)$ xs `++` $(ys$ `++` $zs) = (xs$ `++` $ys)$ `++` zs.

?2. Considerar la siguiente definición de la función `reverse`:

```
reverse ::  [a] -> [a]
reverse [] = []
reverse (x : xs) = reverse xs ++ [x]
```

1. Demostrar $(\forall xs \in$ `[a]`$)(\forall ys \in$ `[a]`$)$ `reverse`$(xs$ `++` $ys) =$ `reverse` ys `++` `reverse` xs.
2. Demostrar, sin usar inducción:
   (a) `reverse` `[`x`]` $=$ `[`x`]`.
   (b) `[`x`]` `++` $xs = x : xs$.
3. Demostrar $(\forall xs \in$ `[a]`$)$ `reverse` (`reverse` $xs) = xs$.

?3. Demostrar $(\forall xs \in$ `[a]`$)(\forall p \in$ `(a -> Bool)`$)$ `takeWhile` $p\,xs$ `++` `dropWhile` $p\,xs = xs$.