Derivación de Programas

Pablo F. Castro

Programación Avanzada, Universidad Nacional de Río Cuarto, Departamento de Computación

2012

Derivación de Programas

La derivación de programas nos permite desarrollar programas **correctos** con respecto a su especificación.

Utilizaremos un conjunto de técnicas para derivar programas utilizando su especificación, las más comunes son:

- Inducción.
- Reemplazo de constantes por variables.
- Modularización.
- Uso de tuplas.
- Generalización por Abstracción.

Usando Inducción para derivar Programas

Podemos usar la inducción matemática para derivar programas. La idea es utilizar la especificación para calcular el programa de la siguiente forma:

- Para los casos bases reemplazamos en la especificación los parámetros de la función con los valores del caso base, y derivamos la implementación.
- Para el caso inductivo, trabajamos con la especificación utilizando como hipotésis que la función a definir funciona correctamente para los valores anteriores al actual.

Un ejemplo, derivemos una implementación para:

$$f.xs = \langle \sum i : 0 \leq i < \#xs : xs.i \rangle$$

Ejemplo de Derivación

Caso Base:

$$
\begin{aligned}
& f.[] \\
& = \text{[Especificación de } f\text{]} \\
& \langle \sum i : 0 \leq i < \#[] : xs.i \rangle \\
& = \text{[Definición de } \#\text{]} \\
& \langle \sum i : 0 \leq i < 0 : xs.i \rangle \\
& = \text{[Aritmética]} \\
& \langle \sum i : \mathit{false} : xs.i \rangle \\
& = \text{[Rango Vacío]} \\
& 0
\end{aligned}
$$

Es decir, hemos derivado: $f.[] = 0$.

Ejemplo de Derivación II

$$
\begin{aligned}
& f.(x \triangleright xs) \\
& = \text{[Especificación de } f\text{]} \\
& \langle \sum i : 0 \leq i < \#(x \triangleright xs) : (x \triangleright xs).i \rangle \\
& = \text{[Definición de } \#\text{]} \\
& \langle \sum i : 0 \leq i < 1 + \#xs : xs.i \rangle \\
& = \text{[Aritmética]} \\
& \langle \sum i : i = 0 \vee 1 \leq i < 1 + \#xs : (x \triangleright xs).i \rangle \\
& = \text{[Partición de Rango y Rango Unitario]} \\
& x + \langle \sum i : 1 \leq i < 1 + \#xs : (x \triangleright xs).i \rangle \\
& = \text{[reemplazando } j + 1 = i \text{]}
\end{aligned}
$$

Ejemplo de Derivación III

$$
\begin{aligned}
& x + \langle \sum j : 1 \leq j + 1 < 1 + \#xs : (x \triangleright xs).(j + 1) \rangle \\
& = \text{[Aritmética]} \\
& x + \langle \sum j : 0 \leq j < \#xs : (x \triangleright xs).(j + 1) \rangle \\
& = \text{[Def. .]} \\
& x + \langle \sum j : 0 \leq j < \#xs : (xs).(j) \rangle \\
& = \text{[Hip.Ind.]} \\
& x + f.xs
\end{aligned}
$$

Cambiando Constantes por Variables

Cuando tenemos constantes en las especificaciones, podemos obtener una especificación más general cambiando estas constantes por variables. Por lo tanto, el problema original es solo un caso particular de la solución general.
Consideremos el siguiente ejemplo:

$$\langle \sum i : 0 \leq i \leq N : X^i \rangle$$

En este caso tenemos dos constantes: X y N. Podemos generalizar esta especificación cambiando N por una variable:

$$\langle \forall n :: f.n = \langle \sum i : 0 \leq i < n : X^i \rangle \rangle$$

Derivemos f. La solución a la especificación original viene dada por $f.N$.

Cambiando Constantes por Variables

Caso base:

$$
\begin{array}{l}
f.0 \\
= \text{[definición]} \\
\langle \sum i : 0 \leq i < 0 : X^i \rangle \\
= \text{[Rango vacío]} \\
0
\end{array}
$$

Caso inductivo:

$$
\begin{array}{l}
f.(n+1) \\
= \text{[definición]} \\
\langle \sum i : 0 \leq i < (n+1) : X^i \rangle \\
= \text{[Aritmetica]} \\
\langle \sum i : 0 \leq i < n \vee n \leq i < n+1 : X^i \rangle \\
= \text{[Aritmetica]}
\end{array}
$$

Cambiando Constantes por Variables (cont)

$$
\begin{aligned}
&\langle \sum i : 0 \leq i < n \vee i = n : X^i \rangle \\
&= \text{[Partición de rango y rango único]} \\
&\langle \sum i : 0 \leq i < n : N^i \rangle + X^n \\
&= \text{[Inducción]} \\
&f.n + X^n
\end{aligned}
$$

Por lo tanto la función queda:

$$
\begin{aligned}
&f : Num \to Num \\
&f.0 \doteq 0 \\
&f.(n+1) \doteq f.n + X^n
\end{aligned}
$$

Si el lenguaje de programación permite calcular X^n hemos terminado, sino debemos implementar una función que la calcule.

Modularización

Cuando un problema es muy complicado, podemos dividirlo en subproblemas, y así facilitar su resolución.

Consideremos el siguiente ejemplo:

$$\forall n :: f.n = \langle \forall i : 0 \leq i < n : X^i \rangle$$

En el caso base obtenemos:

$$f.0 = 0$$

Modularización

En el ejemplo anterior obtuvimos:

$$f.(n+1) = X^n + f.n$$

Si la exponenciación está implementada en el lenguaje de programación terminamos, sino tenemos que introducir una nueva función:

$$f : Num \to Num$$
$$f.0 = 0$$
$$f.(n+1) = g.n + f.n$$
$$[\![g.i = X^i]\!]$$

Ahora derivamos g obtenemos, para el caso base:

$$g.0 = 1$$

Y para el caso inductivo:

Modularización

$$
\begin{aligned}
& g.(n+1) \\
& = \text{Def.} \\
& X * X^n \\
& = \text{Inducción} \\
& X * g.n
\end{aligned}
$$

Es decir, el programa nos queda:

$$
\begin{aligned}
& f : Num \to Num \\
& f.0 = 0 \\
& f.(n+1) = g.n + f.n \\
& \qquad\qquad [\![g.0 = 1 \\
& \qquad\qquad\quad g.(i+1) = X * g.i]\!]
\end{aligned}
$$

Otra derivación...

La función obtenida cuando derivamos puede no ser única. Dividiendo rango de un forma diferente obtendremos otra función:

$$
\begin{aligned}
& f.(n+1) \\
& = \text{[definición]} \\
& \langle \sum i : 0 \leq i < (n+1) : X^i \rangle \\
& = \text{[Aritmetica]} \\
& \langle \sum i : 0 \leq i < 1 \vee 1 \leq i < n+1 : X^i \rangle \\
& = \text{[Partición de rango y Rango único]} \\
& X^0 + \langle \sum i : 1 \leq i < (n+1) : X^i \rangle \\
& = \text{[Reemplazando i por j+1]} \\
& X^0 + \langle \sum j : 1 \leq j+1 < (n+1) : X^{j+1} \rangle \\
& = \text{[Aritmética]} \\
& X^0 + \langle \sum j : 1 \leq j+1 < (n+1) : X^j * X \rangle
\end{aligned}
$$

Otra derivación...

$$
\begin{array}{l}
= \text{[Prop. Cuantificadores]} \\
X^0 + X * \langle \sum j : 1 \leq j + 1 < (n + 1) : X^j \rangle \\
= \text{[Aritmética]} \\
X^0 + X * \langle \sum j : 0 \leq j < n : X^j \rangle \\
= \text{[Inducción]} \\
X^0 + X * f.n
\end{array}
$$

es decir nos queda:

$$
\begin{array}{l}
f : Num \rightarrow Num \\
f.0 = 0 \\
f.(n + 1) = 1 + X * f.n
\end{array}
$$

Uso de Tuplas

Muchas veces podemos utilizar tuplas para mejorar la eficiencia de los programas. Por ejemplo, consideremos la función *fib*:

$$
\begin{aligned}
&fib : Num \to Num \\
&fib.0 = 0 \\
&fib.1 = 1 \\
&fib.(n+2) = fib.(n+1) + fib.n
\end{aligned}
$$

Esta función es exponencial en tiempo: recalcula muchas veces el mismo valor. Tratemos de derivar una función más eficiente.

$$
\begin{aligned}
&g : Num \to (Num, Num) \\
&g.n = (fib.n, fib.(n+1))
\end{aligned}
$$

Uso de Tuplas

Para el caso baso:

$$
\begin{aligned}
& g.0 \\
= & \text{ [Def.g]} \\
& (fib.0, fib.1) \\
= & \text{ [Def.fib]} \\
& (0, 1)
\end{aligned}
$$

Derivemos el caso inductivo:

$$
\begin{aligned}
& g.(n+1) \\
= & \text{ Def. g} \\
& (fib.(n+1), fib.(n+2)) \\
= & \text{ Def. fib} \\
& (fib.(n+1), fib.n + fib.(n+1))
\end{aligned}
$$

Uso de tuplas

La repetición de $fib.(n+1)$ es la causa de la ineficiencia de fib. Usamos definiciones locales para evitar recalcular una expresión.

$$
\begin{array}{l}
= \text{[intro. de } a \text{ y } b\text{]} \\
(fib.(n+1), fib.n + fib.(n+1)) \\
[\![a = fib.n \\
b = fib.(n+1)]\!] \\
= \text{[igualdad de pares]} \\
(fib.(n+1), fib.n + fib.(n+1)) \\
[\![(a,b) = (fib.n, fib.(n+1))]\!] \\
= \text{[reemplazo]} \\
(b, a+b) \\
[\![(a,b) = (fib.n, fib.(n+1))]\!] \\
= \text{[Inducción]}
\end{array}
$$

Uso de tuplas

$$(b, a + b)$$
$$[\![(a, b) = g.n]\!]$$

Es decir:

$$f : Num \to Num$$
$$g.0 = (0, 1)$$
$$g.(n + 1) = (b, a + b)$$
$$[\![(a, b) = g.n]\!]$$

Generalización por Abstracción

Cuando no podemos aplicar la hipotésis inductiva directamente, podemos tratar de de derivar una especificación más general. Por ejemplo:

$$P : [Num] \to Bool$$
$$P.xs = \langle \forall i : 0 \leq i < \#xs : sum.(xs \uparrow i) \geq 0 \rangle$$

Tratar de derivar la implementación de esta función directamente no es factible. Veamos el caso inductivo:

$$P.(x \triangleright xs)$$
$$= \text{[def.]}$$
$$\langle \forall i : 0 \leq i < \#(x \triangleright xs) : sum.((x \triangleright xs) \uparrow i) \geq 0 \rangle$$
$$= \text{[def. } \#\text{]}$$
$$\langle \forall i : 0 \leq i < 1 + \#xs : sum.((x \triangleright xs) \uparrow i) \geq 0 \rangle$$
$$= \text{[part. de rango y lógica]}$$
$$\langle \forall i : 0 \leq i < \#xs : x + sum.(xs \uparrow i) \geq 0 \rangle$$

En este punto no podemos aplicar la hipótesis inductiva!

Generalización (cont)

Una forma de solucionar esto es resolver un problema más general que abarque al problema original:

$$Q : Num \to [Num] \to Bool$$

$$Q.n.xs = \langle \forall i : 0 \leq i < \#xs : n + sum.(xs \uparrow i) \geq 0 \rangle$$

Entonces tenemos: $P.xs = Q.0.xs$. Derivemos el caso inductivo.

$$Q.n.(x \triangleright xs)$$

$$= \text{[Def.]}$$

$$\langle \forall i : 0 \leq i < \#x \triangleright xs : n + sum.((x \triangleright xs) \uparrow i) \geq 0 \rangle$$

$$= \text{[separación de un termino]}$$

$$n \geq 0 \wedge \langle \forall i : 0 \leq i < \#xs : n + sum.(x \triangleright xs) \uparrow (i + 1)) \geq 0 \rangle$$

$$= \text{[def. } \uparrow \text{]}$$

$$n \geq 0 \wedge \langle \forall i : 0 \leq i < \#xs : n + sum.(x \triangleright (xs \uparrow i)) \geq 0 \rangle$$

$$= \text{[Def. de sum]}$$

Generalización (cont)

$$
\begin{array}{l}
n \geq 0 \wedge \langle \forall i : 0 \leq i < \#xs : n + x + sum.(xs \uparrow i) \geq 0 \rangle \\
= \text{[Inducción]} \\
n \geq 0 \wedge Q.(n + x).xs
\end{array}
$$

Es decir, obtenemos:

$$
\begin{array}{l}
Q.n.[] \doteq True \\
Q.n.(x \triangleright xs) \doteq n \geq 0 \wedge Q.(n + x).xs
\end{array}
$$

y:

$$P.xs \doteq Q.0.xs$$