Algoritmos y Estructuras de Datos I - 1º cuatrimestre 2011

Práctico 2: Cuantificación generalizada. Especificación, derivación y verificación de programas.

Docentes: Javier Blanco, Silvia Pelozo, Natalia Bidart, Demetrio Vilela, Walter Alini

Esta guía tiene como objetivos:

- *Consolidar el manejo de expresiones cuantificadas*
- *Obtener las habilidades necesarias para llevar adelante un proceso de derivación o verificación de programas recursivos a partir de especificaciones formales*
- *Completar el proceso de desarrollo llevando los resultados a Haskell*

1. Suponé que $\bigoplus$ es un cuantificador asociado a un operador genérico $\oplus$, que es conmutativo y asociativo (así como el $\forall$ es el cuantificador asociado a la conjunción $\wedge$). Suponé ademas que Z es una constante, y $R.i.j$ y $T.i.j$ predicados arbitrarios (posiblemente dependientes de i y j). Demostrá la siguiente *regla de eliminación de variable* dummy:

$$\langle \bigoplus i, j \ : \ i = Z \wedge R.i.j \ : \ T.i.j \rangle \equiv \langle \bigoplus j \ : \ R.Z.j \ : \ T.Z.j \rangle$$

   a) ¿Es estrictamente necesario que Z sea constante?

   b) El $\exists$ es el cuantificador asociado a la disyunción $\vee$, y éste es asociativo y conmutativo. Por lo tanto, la regla anterior vale para el $\exists$. ¿se puede utilizar esta regla para demostrar $\langle \exists x, y \ : \ x = y \ : \ P.x.y \rangle \equiv \langle \exists x \ : : \ P.x.x \rangle$?

2. Demostrá las siguientes reglas:

   a) *Distributividad de* $\forall$ *respecto a* $\Rightarrow$:

$$\langle \forall i \ : \ R.i \ : \ Z \Rightarrow T.i \rangle \equiv Z \Rightarrow \langle \forall i \ : \ R.i \ : \ T.i \rangle$$

   ¿Qué restricciones se deben establecer sobre Z?

   b) *Instanciación de* $\forall$:

$$\langle \forall i \ : : \ T.i \rangle \Rightarrow T.x$$

   donde x es una variable libre

   ¿Como sería la regla de instanciación para $\exists$? Enunciala y demostrala.

   c) *Intercambio para* $\forall$ *(generalizada)*:

$$\langle \forall i \ : \ R.i \wedge S.i \ : \ T.i \rangle \equiv \langle \forall i \ : \ R.i \ : \ S.i \Rightarrow T.i \rangle$$

3. El cuantificador aritmético N está definido utilizando la sumatoria: $\langle \mathrm{N} i \ : \ R.i \ : \ T.i \rangle \doteq \langle \sum i \ : \ R.i \wedge T.i \ : \ 1 \rangle$

   a) Enunciá y demostrá las reglas de *rango vacío*, *rango unitario* y *partición de rango* para N.

   b) Demostrá $\langle \sum i \ : \ R.i \wedge T.i \ : \ k \rangle = \langle \mathrm{N} i \ : \ R.i \ : \ T.i \rangle \times k$

4. A partir de cada una de las siguientes especificaciones:

   - Dá el tipo de la función
   - Derivá las soluciones algorítmicas correspondientes.
   - Transcribí a Haskell los resultados
   - Compará, si corresponde, los resultados obtenidos con respecto al práctico anterior

   a) $sum\_pares.n = \langle \sum i \ : \ 0 \leq i \leq n \wedge par.i \ : \ i \rangle$

   b) $sum\_cuads.xs = \langle \sum i \ : \ 0 \leq i < \#xs \ : \ xs.i * xs.i \rangle$

c) *cuántos.p.xs* $= \langle \mathrm{N}\ i\ :\ 0 \leq i < \#xs\ :\ p.(xs.i) \rangle$

d) *es_cuadrado.n* $= \langle \exists x\ :\ 0 \leq x \leq n\ :\ x * x = n \rangle$

e) *divide* : $Nat- > Nat- > Bool$, que determina si el primer parámetro divide al segundo.

f) *primo?* : $Nat- > Bool$, que determina si un numero es primo.

g) *listas_iguales* : $[A] \to [A] \to Bool$, que determina si dos listas son iguales, es decir, contienen los mismos elementos en las mismas posiciones respectivamente.

h) *listas_iguales?* : $[A] \to [A] \to Bool$, que determina si dos listas tienen los mismos elementos.

i) *minout* : $[Nat] \to Nat$, que dada una lista de naturales, devuelve el menor número natural que **no** está en la misma.

j) *perm* : $[A] \to [A] \to Bool$, que dada dos listas decide si una es una permutación de la otra.

5. Describí en lenguaje natural y especificá el *tipo* de cada una de las siguientes funciones especificadas formalmente:

a) $f.xs \doteq \langle \mathrm{N}\ i,j\ :\ 0 \leq i < j < \#xs\ :\ xs.i = 0 \wedge xs.j = 0 \rangle$

b) $g.xs \doteq \langle \mathrm{Max}\ p,q\ :\ 0 \leq p < q < \#xs\ :\ xs.p + xs.q \rangle$

c) $h.xs \doteq \langle \mathrm{N}\ k\ :\ 0 \leq k < \#xs\ :\ \langle \forall i\ :\ 0 \leq i < k\ :\ xs.i < xs.k \rangle \rangle$

d) $k.xs \doteq \langle \forall i,j\ :\ 0 \leq i \wedge 0 \leq j \wedge i + j = \#xs - 1\ :\ xs.i = xs.j \rangle$

e) $l.xs \doteq \langle \mathrm{Max}\ p,q\ :\ 0 \leq p \leq q < \#xs \wedge \langle \forall i\ :\ p \leq i < q\ :\ xs.i \geq 0 \rangle\ :\ q - p \rangle$

6. Para cada una de las siguientes funciones:

- Especificá formalmente utilizando cuantificadores
- Derivá soluciones algorítmicas correspondientes
- Transcribí a Haskell los resultados
- Compará, si corresponde, los resultados obtenidos con respecto al práctico anterior

a) *minimo* : $[Int] \to Int$, que calcula el mínimo elemento de una lista de enteros.

b) *iguales* : $[A] \to Bool$, que determina si los elementos de una lista de tipo A son todos iguales entre sí. Suponga que el operador $=$ es la igualdad para el tipo A.

c) *creciente* : $[Int] \to Bool$, que determina si los elementos de una lista de enteros están ordenados en forma creciente.

d) *sum_ant* : $[Num] \to Bool$, que dada una lista de números, determina si algún elemento de la lista es igual a la suma de todos los elementos anteriores dentro de la misma.

e) *pos_min* : $Int \to [Num] \to Bool$, que determina si el n-ésimo elemento de una lista de números contiene al mínimo valor de la misma.

7. Especificá utilizando cuantificadores las funciones *abren* : $[Char] \to Nat$ y *cierran* : $[Char] \to Nat$, que devuelven la cantidad de paréntesis "(" y ")" respectivamente, que ocurren en una lista de caracteres "(" y ")". Luego, *verificá* que las siguientes definiciones recursivas satisfagan esas especificaciones:

$$\begin{array}{rcl} abren.[\,] & \doteq & 0 \\ abren.(\text{`(`} \triangleright xs) & \doteq & 1 + abren.xs \\ abren.(\text{`)`} \triangleright xs) & \doteq & abren.xs \end{array} \qquad \begin{array}{rcl} cierran.[\,] & \doteq & 0 \\ cierran.(\text{`(`} \triangleright xs) & \doteq & cierran.xs \\ cierran.(\text{`)`} \triangleright xs) & \doteq & 1 + cierran.xs \end{array}$$

8. Derivá una función f que compute el cubo de un número natural x, utilizando únicamente sumas. La especificación es muy simple: $f.x = x^3$

Ayuda: Usar inducción y modularización varias veces

9. Dada la siguiente definición recursiva para la funcion *repetir* : $Nat \to Nat \to [Nat]$, que dada una cantidad n y un valor x construye una lista de longitud n con elementos repetidos x:

$$\begin{array}{rcl} repetir.0.x & \doteq & [\,] \\ repetir.(n+1).x & \doteq & x \triangleright repetir.n.x \end{array}$$

Demostrá que es válido $\#repetir.n.x = n$.

10. Expresá utilizando cuantificadores las siguientes sentencias del lenguaje natural:

   a) El elemento x ocurre un número par de veces en la lista xs.

   b) El elemento x ocurre en las posiciones pares de la lista xs.

   c) El elemento x ocurre únicamente en las posiciones pares de la lista xs.

   d) Si x ocurre en la lista xs, entonces y ocurre en alguna posicion anterior en la misma lista.

   e) Existe un elemento de la lista xs que es estrictamente mayor a todos los demás.

   f) Cualquier valor x que anula la funcion f (es decir que $f.x = 0$) ocurre en la lista xs.

   g) En la lista xs solo ocurren valores que anulan la funcion f.

   h) El número natural n es un *cubo perfecto*.

   i) La lista xs es un segmento inicial de la lista ys.

   j) La lista xs es un segmento final de la lista ys.

   k) La lista xs es un segmento de la lista ys.

   l) Las listas xs y ys tienen en común un segmento.

   m) La lista xs posee un segmento **no** inicial y **no** final cuyos valores son mayores a los valores del resto de la misma.

   n) La lista xs de números enteros tiene la misma cantidad de elementos pares e impares.

11. Derivá funciones recursivas a partir de cada una de las especificaciones que escribiste para los ítems a, c, g, h, i, n del ejercicio anterior. Transcribilas en Haskell.

12. Considerá la función *encuentra* que dado un valor de tipo *Int* y una lista de pares *[(Int,String)]* devuelve el segundo componente del par cuyo primer componente es igual al primer parámetro. En el caso que ningún elemento de la lista cumpla con esto, devolver el string vacío. En caso de que haya más de uno, devolver el primero. Por ejemplo:

```
encuentra 10 [(40,"tos"),(10,"uno"),(16,"taza"),(10,"dos")] = "uno"
encuentra 10 [(40,"tos"),(16,"taza"),(10,"dos")] = "dos"
encuentra 102 [(40,"tos"),(103,"vela"),(16,"taza")] = ""
encuentra 102 [] = ""
```

   a) Definí en Haskell la función *encuentra*, sin usar *foldr*. **Ayuda:** Pensar inducción en la longitud de la lista

   b) Definí en Haskell la función utilizando el funcional *foldr*.

   c) Repetí los puntos anteriores, pero teniendo en cuenta que la función debe devolver el último valor en caso de haber más de uno.

13. Mejorá la eficiencia de la función derivada en el ejercicio 8 utilizando la técnica de tuplas.

14. *[Torres de Hanoi]*. Se tienen tres postes numerados 0, 1 y 2, y n discos de distinto tamaño. Inicialmente se encuentran todos los discos ubicados en el poste 0, ordenados según el tamaño con el disco más grande en la base. El problema consiste en llevar todos los discos al poste 2, con las siguientes restricciones:

   a) Se puede mover sólo un disco a la vez

   b) Sólo se puede mover el disco que se encuentra mas arriba en algún poste.

c) No se puede colocar un disco sobre otro de menor tamaño.

Resolvé los siguientes items:

a) Sea $B = \{0, 1, 2\}$. Definí la función $hanoi : B \to B \to B \to Nat \to [(B, B)]$ tal que $hanoi.a.b.c.n$ calcula la secuencia de *movimientos* para llevar n discos desde el poste a hacia el poste c, utilizando posiblemente el poste b de forma auxiliar. Un *movimiento* es un par (B, B) cuya primer componente indica el poste de salida, y la segunda el poste de llegada.

Ayuda: Por ejemplo, *hanoi* para los postes 0, 1 y 2, con dos discos es: $hanoi.0.1.2.2 = [(0, 1),\ (0, 2),\ (1, 2)]$

b) Escribí en Haskell y probá esta función

c) Demostrá $\#hanoi.a.b.c.n = 2^n - 1$

d) ¿En qué movimiento se cambia de poste por primera vez el disco de mayor tamaño?

15. Sea *hanoi* la función que definiste en el ejercicio 14. Derivá una función recursiva para la función $t : B \to B \to B \to Nat \to ([(B, B)],\ [(B, B)],\ [(B, B)])$ especificada como:

$$t.a.b.c.n = (hanoi.a.b.c.n,\ hanoi.b.c.a.n,\ hanoi.c.b.a.n)$$

¿Cuál es el propósito de esta función? Pensá en la eficiencia de *hanoi*.

Ayuda: Puede ser útil calcular manualmente $t.a.b.c.0$, $t.a.b.c.1$ y $t.a.b.c.(n + 2)$

16. Derivá una definición recursiva para la función $f : [Num] \to [Num] \to Num$, que calcula la mínima distancia entre valores de las listas xs y ys, y cuya especificación es la siguiente:

$$f.xs.ys = \langle \text{Min}\, i, j\ :\ 0 \leq i < \#xs \wedge 0 \leq j < \#ys\ :\ |xs.i - ys.j| \rangle$$

Escribí en Haskell y probá esta función.

17. Derivá una definición recursiva para la función $g : [Char] \to [Char] \to Bool$, especificada como sigue:

$$g.xs.ys = \langle \exists as, bs, c, cs\ :\ xs = as +\!\!+ bs \wedge ys = as +\!\!+ (c \triangleright cs)\ :\ bs = [\,] \vee bs.0 < c \rangle$$