DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Algoritmos y Estructuras de Datos

Guia II

Algoritmos y Estructura de Datos

Integrante	LU	Correo electrónico
Cabrera, Juan Elias	501/23	cabreraelias182@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2610 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

2. Funciones auxiliares

2.1. Ejercicio 1

(a) `pred` <u>`raizCuadrada`</u> (x: $\mathbb{Z}$) {

$(\exists c : \mathbb{Z})(c > 0 \ \wedge \ (c * c = x))$

}

(b) `pred` <u>`esPrimo`</u> (x: $\mathbb{Z}$) {

$(x > 1) \wedge (\forall n : \mathbb{Z})((1 < n < x) \longrightarrow_L (x \ mod \ n \neq 0)$

}

2.2. Ejercicio 2

(a) `pred sonCoprimos` (x,y: $\mathbb{Z}$) {

$(\forall n : \mathbb{Z})(((1 < n < x) \wedge (x \ mod \ n = 0)) \longrightarrow_L (y \ mod \ n \neq 0))$

}

(b) `pred mayorPrimoQueDivide` (x,y: $\mathbb{Z}$) {

$(\exists n : \mathbb{Z})((1 < n < x) \wedge esPrimo(n) \wedge (x \ mod \ n = 0) \wedge ((\forall r : \mathbb{Z})((1 < r < x) \wedge esPrimo(r) \wedge (x \ mod \ r = 0)) \longrightarrow_L n \geq r))$

}

2.3. Ejercicio 3

(a) `pred` <u>`todosNaturales`</u> (s: $seq\langle\mathbb{Z}\rangle$) {

$(\forall i : \mathbb{Z})((0 \leq i < |s|) \longrightarrow_L s[i] \geq 0)$

}

(b) `pred` <u>`todosDistintos`</u> (s: $seq\langle T\rangle$) {

$(\forall i : \mathbb{Z})((0 \leq i < |s|) \longrightarrow_L (\forall j : \mathbb{Z})((0 \leq j < |s| \wedge i \neq j) \longrightarrow_L (s[i] \neq s[j])))$

}

2.4. Ejercicio 4

(a) `pred esPrefijo` (s: $seq\langle\mathbb{Z}\rangle$, r: $seq\langle\mathbb{Z}\rangle$) {

$(\forall i : \mathbb{Z})((0 \leq i < |s|) \longrightarrow_L (s[i] = r[i]) \vee ((0 \leq j < |r|) \longrightarrow_L (r[j] = s[j]))$

}

(b) `pred estáOrdenada` (s: $seq\langle\mathbb{Z}\rangle$) {

$(\forall i : \mathbb{Z})(0 \leq i < |s| - 1) \longrightarrow_L (s[i] \leq s[i+1]))$

}

(c) `pred hayUnoParQueDivideAlResto` (s: $seq\langle\mathbb{Z}\rangle$) {

$(\exists i : \mathbb{Z})(((0 \leq i < |s|) \wedge esPar(s[i]) \wedge ((\forall j : \mathbb{Z})(0 \leq j < |s|) \longrightarrow_L (s[j] \ mod \ s[i] = 0)))$

}

(d) `pred enTresPartes` (s: $seq\langle\mathbb{Z}\rangle$) {

TODO

}

2.5. Ejercicio 5

(a) `aux cantApariciones` (e: $\mathbb{Z}$, s: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ $= \sum_{i=0}^{|s|-1}(if\ s[i] = e\ then\ 1\ else\ 0\ fi)$;

(b) `aux sumaIndicesImpares` (s: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ $= \sum_{i=0}^{|s|-1}(if\ (s[i]\ mod\ 2 \neq 0)\ then\ s[i]\ else\ 0\ fi)$;

(c) `aux sumaPositivos` (s: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$ $= \sum_{i=0}^{|s|-1}(if\ (s[i] > 0)\ then\ s[i]\ else\ 0\ fi)$;

(d) `aux sumaInversosMultiplicativos` (s: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{R}$ $= \sum_{i=0}^{|s|-1}(if\ (s[i] \neq 0)\ then\ \frac{1}{s[i]}\ else\ 0\ fi)$;

3. Análisis de especificación

3.1. Ejercicio 6 (Errores marcados en rojo)

(a) `proc progresionGeometricaFactor2` (in l: $seq\langle\mathbb{Z}\rangle$) : Bool

`requiere` $\{True\}$

`asegura` $\{res = True \leftrightarrow ((\forall i : \mathbb{Z})(0 < i < |l| \longrightarrow_L l[i] = 2 * l[i-1]))\}$

(b) `proc minimo` (in l: $seq\langle\mathbb{Z}\rangle$) : Bool

`requiere` $\{True\}$

`asegura` $\{(res \in l) \land (\forall y : \mathbb{Z})(y \in l \cancel{\land y \neq x} \longrightarrow y \geq res)\}$

3.2. Ejercicio 7

(a) `proc indiceDelMaximo` (in l: $seq\langle\mathbb{R}\rangle$) : $\mathbb{Z}$

`requiere` $\{|l| > 0\}$

`asegura` $\{0 \leq res < |l| \land_L ((\forall i :\ ent)(0 \leq i < |l| \longrightarrow_L l[i] \leq l[res])\}$

(I) $l = \langle 1, 2, 3, 4\rangle \rightarrow \langle 3\rangle$

(II) $l = \langle$15.5, -18, 4.215, 15.5, -1$\rangle \rightarrow \langle 0, 3\rangle$

(III) $l = \langle 0, 0, 0, 0, 0, 0\rangle \rightarrow \langle 0, 1, 2, 3, 4, 5\rangle$

(b) `proc indiceDelPrimerMaximo` (in l: $seq\langle\mathbb{R}\rangle$) : $\mathbb{Z}$

`requiere` $\{|l| > 0\}$

`asegura` $\{0 \leq res < |l| \land_L ((\forall i : \mathbb{Z})(0 \leq i < |l| \longrightarrow_L (l[i] < l[res] \lor (l[i] = l[res] \land i \geq res))))\}$

(I) $l = \langle 1, 2, 3, 4\rangle \rightarrow \langle 3\rangle$

(II) $l = \langle$15.5, -18, 4.215, 15.5, -1$\rangle \rightarrow \langle 3\rangle$

(III) $l = \langle 0, 0, 0, 0, 0, 0\rangle \rightarrow \langle 0\rangle$

(c) Para el primero

3.3. Ejercicio 8. *Sea* $f : \mathbb{R} \times \mathbb{R}$ *definida como* :

$$f(x) = \begin{cases} 2b & si\ a \leq 0 \\ b - 1 & en\ otro\ caso \end{cases}$$

(a) `proc f` (in a, b: $\mathbb{R}$) : $\mathbb{R}$

`requiere` $\{True\}$

`asegura` $\{(a < 0 \land res = 2 * b) \land (a \geq 0 \land res = b - 1)\}$

Los problemas en esta especificación son varios, primero, en el asegura se está hablando sobre los valores de entrada lo cual está mal, además, se asegura que res = 2*b y que res = b-1, lo cual no representa lo que quiere nuestra función

(b) `proc f` (in a,b: $\mathbb{R}$) : $\mathbb{R}$

`requiere` $\{True\}$

`asegura` $\{(a < 0 \land res = 2 * b) \lor (a \geq 0 \land res = b - 1)\}$

Acá hay problemas similares, pues de nuevo aseguramos cosas sobre los valores de entrada lo que falseará la función en muchos casos, además, podría pasar que a sea menor a cero pero res no sea 2*b, lo cual de nuevo falsearía nuestra especificación

(c) `proc f` (in a,b: $\mathbb{R}$) : $\mathbb{R}$

`requiere` $\{True\}$

`asegura` $\{(a < 0 \longrightarrow res = 2 * b) \lor (a \geq 0 \longrightarrow res = b - 1)\}$

En este caso, el problema es que nosotros aseguramos que se cumple una implicancia o la otra, y no tiene por qué ser así, no hacemos que res dependa de a, si no que damos a entender que puede pasar algo de lo que no estamos seguros.

(d) `proc f` (in a,b: $\mathbb{R}$) : $\mathbb{R}$

`requiere` $\{True\}$

`asegura` $\{res = (if\ a < 0\ then\ 2 * b\ else\ b - 1\ fi)\}$

Esta es la especificación correcta pues es la única que nos garantiza que res depende del valor de a.

3.4. Ejercicio 9

`proc unoMasGrande` (in x: $\mathbb{R}$) : $\mathbb{R}$
`requiere` $\{True\}$
`asegura` $\{res > x\}$

(a) El algoritmo devuelve 9, y este resultado hace verdadera la postcondición pues 9 ¿3

(b) La únicas de estas entradas que cumplen la postcondición son las negativas, dado que al elevarlas al cuadrado se hacen positivas.

(c) Una posible precondición sería la siguiente: $(x > 1) \lor (x < 0)$

4. Relación de fuerza

4.1. Ejercicio 10

$P1 = x \leq 0 \quad Q1 = r \geq x^2$
$P2 = x \leq 10 \quad Q2 = r \geq 0$
$P3 = x \leq -10 \quad Q3 = r = x^2$

a) P3 $\longrightarrow$P2 $\longrightarrow$P1

b) Q3 $\longrightarrow$Q1 $\longrightarrow$Q2

c) Hecho

d)
   a) Sí, pues $x \leq -10$ es más fuerte que $x \leq 0$
   b) No se puede asegurar pues el algoritmo sabes que cumple para x menores a 0, no sabemos nada de los numeros entre 0 y 10.
   c) Sí, pues $r \geq x^2$ es más fuerte que $r \geq 0$
   d) No necesariamente, pues el requiere es el mismo pero, por ejemplo, si defino $x = -1$, un res válido podría ser 10, el cual es distinto de x^2
   e) No pues si bien el requiere es más fuerte que el original, el asegura no lo es, por ejemplo, $x = 0{,}5$ un valor de res válido podría ser 2, pero para el item e), 0,2 sería un valor válido, siendo $0{,}7 < (0{,}5)^2$
   f) No porque este requiere es más débil que el original, así que no podemos garantizar nada acerca de que el algoritmo funcione para esta espeficicación.

e) Para que sea seguro reemplazar las especificaciones, necesitamos que nuestro requiere y/o asegura nuevos, sean más fuertes que los originales, así podríamos decir que los originales se deducen de los nuevos.

4.2. Ejercicio 11

proc p1 (in x: $\mathbb{R}$, in n: $\mathbb{Z}$) : $\mathbb{Z}$
  requiere $\{x \neq 0\}$
  asegura $\{x^n - 1 < res \leq x^n\}$

proc p2 (in x: $\mathbb{R}$, in n: $\mathbb{Z}$) : $\mathbb{Z}$
  requiere $\{n \leq 0 \longrightarrow x \neq 0\}$
  asegura $\{res = \lfloor x^n \rfloor\}$

a) Si x y n cumplen la precondición de p1, para que cumplan la precondición de p2, debería pasar que p1 sea más fuerte que p2.
Para demostrarlo, deberíamos ver que $(x \neq 0) \longrightarrow (n \leq 0 \longrightarrow x \neq 0)$ es una tautología. Podemos hacerlo con una tabla de verdad.

$(x \neq 0)$	$(n \leq 0)$	$(n \leq 0 \longrightarrow x \neq 0)$	$(x \neq 0) \longrightarrow (n \leq 0 \longrightarrow x \neq 0)$
V	V	V	V
V	F	V	V
F	V	F	V
F	F	V	V

Como podemos apreciar, demostramos que si x y n hacen verdadera la precondición de p1, también hacen verdadera la precondición de p2.

b) Para esto, vamos a hacer otra tabla de verdad

$(res = \lfloor x^n \rfloor)$	$(x^n - 1 < res \leq x^n)$	$(res = \lfloor x^n \rfloor) \longrightarrow (x^n - 1 < res \leq x^n)$
V	V	V
V	F	F
F	V	F
F	F	V

Acá el único caso que nos interesa analizar es cuando alpha es V y beta es F, debemos analizar si es posible que $x^n - 1 < res \leq x^n$ sea falso sabiendo que $res = \lfloor x^n \rfloor$. Efectivamente, es imposible esto, pues res al ser $\lfloor x^n \rfloor$, es más grande siempre que $x^n - 1$. Y el único caso en el que la desigualdad se incumple es cuando $res = x^n - 1$, pero este caso no tiene sentido. Concluimos, entonces, que alpha es más fuerte que beta. Por lo tanto, tenemos que la postcondición de p2 implica la postcondición de p1, por lo tanto, si, será verdadera la postcondición de p1 con este valor de res.

c) Como pudimos demostrar que esto funcionaba para un valor x y n genéricos, podemos asegurar que a satisface p1.

5. Especificación de problemas

5.1. Ejercicio 12. Especificar.

a) `proc esPar` (in n: $\mathbb{Z}$) : Bool

`requiere` $\{True\}$

`asegura` $\{result = true \leftrightarrow (n\ mod\ 2 = 0)\}$

b) `proc esMultiplo` (in n,m: $\mathbb{Z}$) : Bool

`requiere` $\{True\}$

`asegura` $\{result = true \leftrightarrow (m\ mod\ n = 0)\}$

`pred todosSonDivisoresNaturalesYPositivos` (s: $seq\langle\mathbb{Z}\rangle$, n: $\mathbb{Z}$) {

$(\forall m : \mathbb{Z})(m \in s \longrightarrow (n\ mod\ m = 0\ \wedge (m > 0))$

}

`pred noFaltaNinguno` (s: $seq\langle\mathbb{Z}\rangle$, n: $\mathbb{Z}$) {

$(\forall m : \mathbb{Z})(n\ mod\ m = 0 \wedge m > 0) \longrightarrow m \in s))$

}

`pred todosSonPrimos` (s: $seq\langle\mathbb{Z}\rangle$) {

$(\forall m : \mathbb{Z})(m \in s \longrightarrow (esPrimo(m))$

}

c) `proc listarDivisoresPositivos` (in n: $\mathbb{Z}$) : $seq\langle\mathbb{Z}\rangle$

`requiere` $\{n \neq 0\}$

`asegura` $\{todosSonDivisoresNaturalesYPositivos(result, n) \wedge noHayRepetidos(result) \wedge noFaltaNinguno(result, n)\}$

`pred esPotenciaDePrimoMasGrandeQueDivide` (in n, m, p: $\mathbb{Z}$) {

$(n\ mod\ p^m = 0) \longrightarrow (n\ mod\ p^{m+1} \neq 0)$

}

d) `proc` `descomponerEnPrimos` (in n: $\mathbb{Z}$) : $seq\langle \mathbb{Z} \times \mathbb{Z}\rangle$

`requiere` $\{n > 1\}$

`asegura` $\{(\forall i : \mathbb{Z})(0 \leq i < |result| \longrightarrow esPrimo(result[i]_0)$
$\wedge\ (todosSonDivisoresNaturalesYPositivos(result[i]_0, n))$
$\wedge\ (esPotenciaDePrimoMasGrandeQueDivide(n, result[i]_1, result[i]_0)))$
$\wedge\ ((\forall m : \mathbb{Z})(0 \leq m < |result| - 1 \longrightarrow_L result[m]_0 \leq result[m + 1]_0))\}$

5.2. Ejercicio 13. Especificar.

a) `proc` `estaIncluida` (s,t: $seq\langle T\rangle$) : Bool

`requiere` $\{True\}$

`asegura` $\{(\forall i : \mathbb{Z})(0 \leq i < |s| \longrightarrow_L (\exists j : \mathbb{Z})(0 \leq j < |t| \wedge s[i] = s[t]))\}$

b) `proc` `interseccion` (s,t: $seq\langle T\rangle$) : $seq\langle T\rangle$

`requiere` $\{True\}$

`asegura` $\{(\forall n : T)((n \in s \wedge n \in t) \leftrightarrow (n \in result)) \wedge (\forall m : T)(m \in result \longrightarrow cantApariciones(m, result) = if\ cantApariciones(m, s) < cantApariciones(m, t)$
$then\ cantApariciones(m, s)\ else\ cantApariciones(m, t)\ fi) \wedge (|result| \leq (if\ |s| \geq |t|\ then\ |t|\ else\ |s|\ fi)\}$

`aux` `aCuantosDivide` (n: $\mathbb{Z}$, s: $seq\langle \mathbb{Z}\rangle$) : $\mathbb{Z}$ $= \sum_{i=0}^{|s|-1}(if\ (s[i]\ mod\ n = 0)\ then\ 1\ else\ 0\ fi)$ **;**

c) `proc` `elQueMasDivide` (s: $seq\langle \mathbb{Z}\rangle$) : $\mathbb{Z}$

`requiere` $\{|s| > 0\}$

`asegura` $\{(result \in s) \wedge ((\exists m : \mathbb{Z})(m \in s \wedge m\ mod\ result = 0) \wedge (\neg(\exists n : \mathbb{Z})(n \in s \wedge (aCuantosDivide(n, s) > aCuantosDivide(result, s))\}$

d) `proc` `secuenciaConValorMaximo` (s: $seq\langle seq\langle \mathbb{Z}\rangle\rangle$) : $seq\langle \mathbb{Z}\rangle$

`requiere` $\{True\}$

`asegura` $\{(result \in s) \wedge ((\exists n : \mathbb{Z})(n \in result \wedge (\forall i, j : \mathbb{Z})(0 \leq i < |s| \wedge 0 \leq j < |s[i]|)(n \geq s[i][j])))))\}$

`pred` `apareceEnEseOrden` (s,t: $seq\langle T\rangle$) {

$estaIncluida(s, t) \wedge ((\forall i : \mathbb{Z})(0 \leq i < |s| - 1 \wedge (\exists j : \mathbb{Z})(0 \leq j < |t| - 1 \wedge s[i] = t[j]) \longrightarrow_L (s[i + 1] = t[j + 1]))$

}

e) `proc` `secuenciaDePartes` (in l: $seq\langle T\rangle$) : $seq\langle seq\langle T\rangle\rangle$

`requiere` $\{todosDistintos(l)\}$

`asegura` $\{(|result| = 2^{|l|}) \wedge ((\forall r : seq\langle T\rangle)(estaIncluida(r, l) \leftrightarrow (r \in result$
$\wedge\ (apareceEnEseOrden(r, l)))\}$

6. Especificación de problemas usando inout

6.1. Ejercicio 14

a) `proc` `sumar` (inout a, b, c: $\mathbb{Z}$)

`requiere` $\{True\}$

asegura $\{a + b = c\}$

Esta no es correcta puesto que "modifica.ª y b, de forma rara, declarando que su suma es igual a c, lo cual no tiene sentido porque queremos devolver c siendo la suma

b) `proc sumar` (in a, b: $\mathbb{Z}$, inout c: $\mathbb{Z}$)

  `requiere` $\{True\}$

  `asegura` $\{c = a + b\}$

Esta es la correcta pues declaramos a y b como variables in pues no las modificamos nunca, y decimos que la variable inout c es modificada al valor de la suma entre a y b.

c) `proc sumar` (in a, b: $\mathbb{Z}$, inout c: $\mathbb{Z}$)

  `requiere` $\{a = A_0 \wedge b = B_0\}$

  `asegura` $\{a = A_0 \wedge b = B_0 \wedge c = a + b\}$

Esta cumple lo pedido pero especifica cosas innecesarias, aunque cumple lo que se pide.

6.2. Ejercicio 15

1. `proc tomarPrimero` (inout l: $seq\langle\mathbb{R}\rangle$) : $\mathbb{R}$

  `requiere` $\{|l| > 0\}$

  `asegura` $\{res = head(l)\}$

Esta no es correcta pues si bien devuelve el primer elemento de l, no la modifica, que es lo que pide el enunciado.

2. `proc tomarPrimero` (inout l: $seq\langle\mathbb{R}\rangle$) : $\mathbb{R}$

  `requiere` $\{|l| > 0 \wedge l = L_0\}$

  `asegura` $\{res = head(L_0)\}$

Ocurre el mismo problema, nunca modificamos l

3. `proc tomarPrimero` (inout l: $seq\langle\mathbb{R}\rangle$) : $\mathbb{R}$

  `requiere` $\{|l| > 0\}$

  `asegura` $\{res = head(L_0) \wedge |l| = |L_0| - 1\}$

El problema acá es que si bien si hacemos todo bien, debe pasar lo de la longitud de l, si no indicamos qué es l ahora, estaríamos dando cualquier opción con un elemento menos que antes como válida.

4. `proc tomarPrimero` (inout l: $seq\langle\mathbb{R}\rangle$) : $\mathbb{R}$

  `requiere` $\{|l| > 0 \wedge l = L_0\}$

  `asegura` $\{res = head(L_0) \wedge l = tail(L_0)\}$

Esta es la solución adecuada pues devolvemos el valor adecuado, y además decimos que l es la cola (equivalente a sacar su primer elemento)

6.3. Ejercicio 16

a) `proc duplicarPares` (inout l: $seq\langle\mathbb{Z}\rangle$)

   `requiere` $\{l = L_0\}$

   `asegura` $\{(|l| = |L_0|) \land (\forall i : \mathbb{Z})(0 \leq i < |l| \land i\ mod\ 2 = 0) \longrightarrow_L l[i] = 2 * L_0[i])\}$

   Mientras en un proc con variable inout no modifiquemos la variable, deberíamos referirnos a ella como old() o haciendo un cambio de variable en el requiere. Acá lo hace, pero a la hora de ver si es un índice válido, debería poner que es índice válido de L_0, además de que no damos detalles de las posiciones impares.

b) `proc duplicarPares` (inout l: $seq\langle\mathbb{Z}\rangle$)

   `requiere` $\{l = L_0\}$

   `asegura` $\{(\forall i : \mathbb{Z})(0 \leq i < |l| \land i\ mod\ 2 \neq 0) \longrightarrow_L l[i] = L_0[i])$
   $\land (\forall i : \mathbb{Z})(0 \leq i < |l| \land i\ mod\ 2 = 0) \longrightarrow_L l[i] = 2 * L_0[i])\}$

   Acá tenemos el mismo problema que en el anterior en los índices.Además, no aclaramos que deben tener la misma longitud l y L_0

c) `proc duplicarPares` (inout l: $seq\langle\mathbb{Z}\rangle$)

   `requiere` $\{True\}$

   `asegura` $\{|l| = |res| \land (\forall i : \mathbb{Z})((0 \leq i < |l| \land i\ mod\ 2 \neq 0) \longrightarrow_L res[i] = l[i]) \land (\forall i : \mathbb{Z})((0 \leq i < |l| \land i\ mod\ 2 = 0) \longrightarrow_L res[i] = 2 * l[i])\}$

   Acá estamos devolviendo una lista que cumple lo pedido pero no hacemos que se modifique l.

   Alternativa correcta

   `proc duplicarPares` (inout l: $seq\langle\mathbb{Z}\rangle$)

   `requiere` $\{l = L_0\}$

   `asegura` $\{((\forall i : \mathbb{Z})(0 \leq i < |L_0| \land i\ mod\ 2 = 0) \longrightarrow_L l[i] = 2 * L_0[i])$
   $\land ((\forall j : \mathbb{Z})(0 \leq j < |L_0| \land i\ mod\ 2 \neq 0 \longrightarrow_L l[i] = L_0[i])) \land (|l| = |L_0|)\}$

6.4. Ejercicio 17

`pred esPrimoHermano` (p,n: $\mathbb{Z}$) {
  $\neg(\exists m : \mathbb{Z})(esPrimo(m) \land m > p)$
}

a) `proc primosHermanos` (inout l: $seq\langle\mathbb{Z}\rangle$)

   `requiere` $\{todosSonMayoresADos(l) \land l = L_0\}$

   `asegura` $\{(\forall i : \mathbb{Z})(0 \leq i < L_0 \longrightarrow_L esPrimoHermano(l[i], L_0[i]) \land (|l| = |L_0|)\}$

b) `proc reemplazar` (inout l: $seq\langle\mathsf{Char}\rangle$, in a,b: Char)

   `requiere` $\{l = L_0\}$

   `asegura` $\{(\forall i : \mathbb{Z})(0 \leq i < |L_0| \land L_0[i] = a \longrightarrow_L l[i] = b) \land (|l| = |L_0|) \land (\forall j : \mathbb{Z})(0 \leq j < |L_0| \land L_0[j] \neq a \longrightarrow_L l[j] = L_0[j])\}$

c) `proc limpiarDuplicados` (inout l: $seq\langle\mathsf{Char}\rangle$, out dup: $seq\langle\mathsf{Char}\rangle$)

   `requiere` $\{l = L_0\}$

`asegura` $\{((c \in L_0 \land cantApariciones(c, L_0) > 1) \leftrightarrow c \in out)$
$\land\ (\forall c_1 : \mathsf{Char})(c_1 \in out \longrightarrow (cantApariciones(c_1, out) = cantApariciones(c_1, L_0) - 1)$
$\land\ ((\forall c_2 : \mathsf{Char})(c_2 \in l \land c_2 \in L_0 \longrightarrow_L (cantApariciones(c_2, l) = 1)))$
$\land\ (\forall c_3 : \mathsf{Char})(c_3 \in L_0 \land cantApariciones(c_3, L_0) \geq 1 \longrightarrow_L c \in l)\}$