Algoritmos y Estructuras de Datos

Guía Práctica 2
Especificación de problemas

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

2.1. Funciones auxiliares

Ejercicio 1. Nombrar los siguientes predicados sobre enteros:

a) `pred ????` (x: $\mathbb{Z}$) {
$(\exists c : \mathbb{Z})(c > 0 \land (c * c = x))$
}

b) `pred ????` (x: $\mathbb{Z}$) {
$(\forall n : \mathbb{Z})((1 < n < x) \rightarrow_L (x \mod n \neq 0))$
}

Solución

a) *esCuadrado*

b) *esPrimo*

Ejercicio 2. Escriba los siguientes predicados sobre números enteros en lenguaje de especificación:

a) pred *sonCoprimos* $(x, y : \mathbb{Z})$ que sea verdadero si y sólo si x e y son coprimos.

b) pred *mayorPrimoQueDivide* $(x : \mathbb{Z}, y : \mathbb{Z})$ que sea verdadero si y es el mayor primo que divide a x.

Solución

a) `pred sonCoprimos` (x, y: $\mathbb{Z}$) {
$(\forall z : \mathbb{Z})(2 \leq z \leq \min(x, y) \rightarrow_L ((x \mod z) = 0 \longleftrightarrow (y \mod z) \neq 0))$
}

b) `pred mayorPrimoQueDivide` (x, y: $\mathbb{Z}$) {
$esPrimo(y) \land_L x \mod y = 0 \land (\forall p : \mathbb{Z})((esPrimo(p) \land_L x \mod p = 0) \rightarrow_L p \leq y)$
}

Ejercicio 3. Nombre los siguientes predicados auxiliares sobre secuencias de enteros:

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

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

}

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

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

}

Solución

a) *todosPositivos*

b) *sinRepetidos*

Ejercicio 4. Escriba los siguientes predicados auxiliares sobre secuencias de enteros, aclarando los tipos de los parámetros que recibe:

a) *esPrefijo*, que determina si una secuencia es prefijo de otra.

b) *estáOrdenada*, que determina si la secuencia está ordenada de menor a mayor.

c) *hayUnoParQueDivideAlResto*, que determina si hay un elemento par en la secuencia que divide a todos los otros elementos de la secuencia.

d) *enTresPartes*, que determina si en la secuencia aparecen (de izquierda a derecha) primero 0s, después 1s y por último 2s. Por ejemplo $\langle 0, 0, 1, 1, 1, 1, 2\rangle$ cumple con *enTresPartes*, pero $\langle 0, 1, 3, 0\rangle$ o $\langle 0, 0, 0, 1, 1\rangle$ no. ¿Cómo modificaría la expresión para que se admitan cero apariciones de 0s, 1s y 2s (es decir, para que por ejemplo $\langle 0, 0, 0, 1, 1\rangle$ o $\langle\rangle$ sí cumplan *enTresPartes*)?

Solución

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

$|s| \leq |t| \land_L (\forall i : \mathbb{Z})((0 \leq i < |s|) \rightarrow_L (s[i] = t[i]))$

}

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

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

}

c) `pred esPar` (n: $\mathbb{Z}$) {

$n \mod 2 = 0$

}

`pred divideA` (n: $\mathbb{Z}$, m: $\mathbb{Z}$) {

$n \neq 0 \land_L m \mod n = 0$

}

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

$(\exists i : \mathbb{Z})((0 \leq i < |s| \land_L esPar(s[i])) \land_L ((\forall j : \mathbb{Z})((0 \leq j < |s|) \rightarrow_L divideA(s[i], s[j]))))$

}

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

$(\exists n : \mathbb{Z})((n \in s \wedge esPar(n)) \wedge ((\forall m : \mathbb{Z})(m \in s \rightarrow divideA(n, m))))$

}

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

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

}

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

$estáOrdenada(s) \wedge sóloCeroUnoYDos(s)$

}

donde

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

$(\forall i : \mathbb{Z})(0 \leq i < |s|) \rightarrow_L (s[i] = 0 \vee s[i] = 1 \vee s[i] = 2) \wedge$
$(\exists i : \mathbb{Z})(0 \leq i < |s|) \wedge_L (s[i] = 0) \wedge$
$(\exists i : \mathbb{Z})(0 \leq i < |s|) \wedge_L (s[i] = 1) \wedge$
$(\exists i : \mathbb{Z})(0 \leq i < |s|) \wedge_L (s[i] = 2)$

}

Extra. Bastaría con tener

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

$(\forall i : \mathbb{Z})(0 \leq i < |s|) \rightarrow_L (s[i] = 0 \vee s[i] = 1 \vee s[i] = 2)$

}

Ejercicio 5. Sea s una secuencia de elementos de tipo $\mathbb{Z}$. Escribir una expresión (utilizando sumatoria y productoria) tal que:

a) Cuente la cantidad de veces que aparece el elemento e de tipo $\mathbb{Z}$ en la secuencia s.

b) Sume los elementos en las posiciones impares de la secuencia s.

c) Sume los elementos mayores a 0 contenidos en la secuencia s.

d) Sume los inversos multiplicativos ($\frac{1}{x}$) de los elementos contenidos en la secuencia s distintos a 0.

Solución

a) $apariciones = \sum_{i=0}^{|s|-1} \texttt{IfThenElse}(s[i] = e,\ 1,\ 0)$

b) $sumaPosPares = \sum_{i=0}^{|s|-1} \texttt{IfThenElse}(i \mod 2 \neq 0,\ s[i],\ 0)$

c) $sumaPositivos = \sum_{i=0}^{|s|-1} \texttt{IfThenElse}(s[i] > 0,\ s[i],\ 0)$

d) $sumaInversos = \sum_{i=0}^{|s|-1} \texttt{IfThenElse}(s[i] \neq 0,\ \frac{1}{s[i]},\ 0)$

2.2. Análisis de especificación

Ejercicio 6. Las siguientes especificaciones no son correctas. Indicar por qué y corregirlas para que describan correctamente el problema.

a) `progresionGeometricaFactor2`: Indica si la secuencia l representa una progresión geométrica factor 2. Es decir, si cada elemento de la secuencia es el doble del elemento anterior.

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

requiere $\{True\}$

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

b) `minimo`: Devuelve en res el menor elemento de l.

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

requiere $\{True\}$

asegura $\{(\forall y : \mathbb{Z})((y \in l \wedge y \neq x) \rightarrow y > res)\}$

Solución

a) Hay problemas de bordes. Cuando $i = 0$, $l[i-1]$ se indefine

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

requiere $\{True\}$

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

b) El antecedente de la implicación no tiene relación con el consecuente. La lista debería tener al menos un elemento, si no se indefine. No garantiza que la salida esté contenida en la secuencia de entrada.

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

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

asegura $\{res \in l \wedge (\forall y : \mathbb{Z})(y \in l \rightarrow_L y \geq res)\}$

Ejercicio 7. Para los siguientes problemas, dar todas las soluciones posibles a las entradas dadas:

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

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

asegura $\{0 \leq res < |l| \wedge_L ((\forall i : \mathbb{Z})(0 \leq i < |l| \rightarrow_L l[i] \leq l[res])\}$

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

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

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

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

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

asegura $\{0 \leq res < |l| \wedge_L ((\forall i : \mathbb{Z})(0 \leq i < |l| \rightarrow_L (l[i] < l[res] \vee (l[i] = l[res] \wedge i \geq res))))\}$

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

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

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

c) ¿Para qué valores de entrada indiceDelPrimerMaximo y indiceDelMaximo tienen necesariamente la misma salida?

Solución

a) I) $l = \langle 1, 2, 3, 4\rangle \longrightarrow 3$

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

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

b) I) $l = \langle 1, 2, 3, 4\rangle \longrightarrow 3$

II) $l = \langle 15.5, -18, 4.215, 15.5, -1\rangle \longrightarrow 0$

III) $l = \langle 0, 0, 0, 0, 0, 0\rangle \longrightarrow 0$

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

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

Indicar cuáles de las siguientes especificaciones son correctas para el problema de calcular $f(a, b)$. Para aquellas que no lo son, indicar por qué.

a) `proc f` (in a, b: $\mathbb{R}$) : $\mathbb{R}$
   `requiere` $\{True\}$
   `asegura` $\{(a < 0 \land res = 2 * b) \land (a \geq 0 \land res = b - 1)\}$

b) `proc f` (in a, b: $\mathbb{R}$) : $\mathbb{R}$
   `requiere` $\{True\}$
   `asegura` $\{(a < 0 \land res = 2 * b) \lor (a \geq 0 \land res = b - 1)\}$

c) `proc f` (in a, b: $\mathbb{R}$) : $\mathbb{R}$
   `requiere` $\{True\}$
   `asegura` $\{(a < 0 \to res = 2 * b) \lor (a \geq 0 \to res = b - 1)\}$

d) `proc f` (in a, b: $\mathbb{R}$) : $\mathbb{R}$
   `requiere` $\{True\}$
   `asegura` $\{res = \texttt{IfThenElse}(a < 0,\ 2 * b,\ b - 1)\}$

Solución

a) Es **incorrecta**. No se cumple nunca. Se puede ver que es equivalente a $(P(a) \land Q(b)) \land (\neg P(a) \land R(b)) \equiv (P(a) \land \neg P(a)) \land Q(b) \land R(b)$ que es una contradicción.

b) Es **correcta**. Es la forma de desglosar un condicional (if).

c) Es **incorrecta**. Es siempre verdadera. Cuando el antecedente de uno es falso la implicación es verdadera y es trivial notar que a es $a < 0$ o $a \geq 0$.

d) Es **correcta**. Evalúa una condición que depende de a. La rama verdadera ($a < 0$) retorna $2 * b$, mientras que la rama falsa ($a \geq 0$) retorna $b - 1$. En ambos casos se asigna un término a la variable res de salida que coincide con lo mostrado para la función partida f.

Ejercicio 9. Considerar la siguiente especificación, junto con un algoritmo que dado x devuelve x^2.

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

a) ¿Qué devuelve el algoritmo si recibe $x = 3$? ¿El resultado hace verdadera la postcondición de unoMasGrande?

b) ¿Qué sucede para las entradas $x = 0{,}5$, $x = 1$, $x = -0{,}2$ y $x = -7$?

c) Teniendo en cuenta lo respondido en los puntos anteriores, escribir una **precondición** para unoMasGrande, de manera tal que el algoritmo cumpla con la especificación

Solución

a) Devuelve 9 y hace verdadera la postcondición.

b) Devuelven 0,25, 1, 0,04 y 49 respectivamente.

c) `requiere` $\{|x| > 1\}$

2.3. Relación de fuerza

Ejercicio 10. Sean x y r variables de tipo $\mathbb{R}$. Considerar los siguientes predicados:

P1: $\{x \leq 0\}$
P2: $\{x \leq 10\}$
P3: $\{x \leq -10\}$

Q1: $\{r \geq x^2\}$
Q2: $\{r \geq 0\}$
Q3: $\{r = x^2\}$

a) Indicar la relación de fuerza entre P1, P2 y P3

b) Indicar la relación de fuerza enrte Q1, Q2 y Q3

c) Escribir 2 programas que cumplan con la siguiente especificación:

   `proc hagoAlgo` (in x: $\mathbb{R}$) : $\mathbb{R}$
     `requiere` $\{x \leq 0\}$
     `asegura` $\{res \geq x^2\}$

d) Sea A un algoritmo que cumple con la especificación del ítem anterior. Decidir si necesariamente cumple las siguientes especificaciones:

   a) `requiere` $\{x \leq -10\}$, `asegura` $\{r \geq x^2\}$
   b) `requiere` $\{x \leq 10\}$, `asegura` $\{r \geq x^2\}$
   c) `requiere` $\{x \leq 0\}$, `asegura` $\{r \geq 0\}$
   d) `requiere` $\{x \leq 0\}$, `asegura` $\{r = x^2\}$
   e) `requiere` $\{x \leq -10\}$, `asegura` $\{r \geq 0\}$
   f) `requiere` $\{x \leq 10\}$, `asegura` $\{r = x^2\}$

e) ¿Qué conclusión pueden sacar? ¿Qué debe cumplirse con respecto a las precondiciones y postcondiciones para que sea seguro **reemplazar la especificación**?

Solución

a) $P3$ es más fuerte que $P1$ y $P2$. $P1$ es más fuerte que $P2$

$P3 \rightarrow P1$, $P3 \rightarrow P2$, $P1 \rightarrow P2$

b) $Q3$ es más fuerte que $Q1$ y $Q2$. $Q1$ es más fuerte que $Q2$

$Q3 \rightarrow Q1$, $Q3 \rightarrow Q2$, $Q1 \rightarrow Q2$

c) *a*) La precondición nueva es más fuerte que la original, por lo tanto alcanza para que ambos algoritmos no se indefinan. La poscondición no cambia.

b) Como la precondición es más débil que la original, no podemos asegurar que el algoritmo vaya a funcionar con los valores de entrada. De hecho, el algoritmo $hagaAlgo2$ se indefine cuando recibe valores entre 0 y 10.

c) La poscondición es más débil que la original, por lo que si nuestros algoritmos cumplían las condiciones originales, deberían cumplir estas también.

d) La poscondición es más fuerte que la original, no tenemos garantía de que se cumpla para los algoritmos propuestos. De hecho, $hagaAlgo2$ no cumple con la poscondición propuesta. Si $x = 0$, la salida de la función es $hagaAlgo2(0) = 2$ y se ve que $0 \neq 2$.

e) Como la precondición es más fuerte y la poscondición más débil, tenemos garantía que se cumplirán las condiciones dispuestas para cualquier algoritmo que cumpliera con lo pedido en el item anterior.

f) En este caso ocurre todo lo contrario, la precondición es más débil y la poscondición más fuerte, por lo que no tenemos ningún tipo de garantía. Es decir, no sabemos si la salida cumplirá con lo pedido y no tenemos garantía de que nos alcance con los requisitos exigidos sobre la entrada.

d) La **precondición** debería ser más restrictiva, es decir, debería ser más fuerte que la original. Ya que así garantizo que se cumplan los criterios mínimos necesarios para que mi programa no se indefina ni se rompa.

La **poscondición** debe ser más laxa, es decir, debería ser más débil que la original. Ya que así garantizo que todas las salidas posibles cumplen también con el criterio original.

Ejercicio 11. Considerar las siguientes dos especificaciones, junto con un algoritmo a que satisface la especificación de `p2`.

`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` $\{n \leq 0 \rightarrow x \neq 0\}$
    `asegura` $\{res = \lfloor x^n \rfloor\}$

a) Dados valores de x y n que hacen verdadera la precondición de `p1`, demostrar que hacen también verdadera la precondición de `p2`.

b) Ahora, dados estos valores de x y n, supongamos que se ejecuta a: llegamos a un valor de res que hace verdadera la postcondición de `p2`. ¿Será también verdadera la postcondición de `p1` con este valor de res?

c) ¿Podemos concluir que a satisface la especificación de `p1`?

Solución

a) El consecuente de la precondición de `p2` es la pre de `p1`, por lo tanto, sin importar el valor de n, si la pre de `p1` es verdadera $\Rightarrow$ también lo es la de `p2`

También podemos pensarlo como dos casos. $n \leq 0$ y $n > 0$. La precondición de `p2` requiere que en el primer caso $x \neq 0$. Basados en la precondición de `p1`, vemos que esto se cumple para ese caso como para cuando $n > 0$.

b) Sí, $x^n - 1 < \lfloor x^n \rfloor \leq x^n$ por la definición de $\lfloor x \rfloor$.

c) Sí, ya que cumple con las condiciones planteadas en el ítem **8.d**.

2.4. Especificación de problemas

Ejercicio 12. Especificar los siguientes problemas:

a) Dado un entero, decidir si es par

b) Dado un entero n y otro m, decidir si n es un múltiplo de m

c) Dado un entero, listar todos sus divisores positivos (sin duplicados)

d) Dado un entero positivo, obtener su descomposición en factores primos. Devolver una secuencia de tuplas (p, e), donde p es un factor primo y e es su exponente, ordenada en forma creciente con respecto a p

Solución

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

    `requiere` $\{True\}$

    `asegura` $\{res = True \leftrightarrow x \mathsf{\ mod\ } 2 = 0\}$

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

    `requiere` $\{True\}$

    `asegura` $\{res = True \leftrightarrow (\exists p : \mathbb{Z})(n \times p = m)\}$

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

    `requiere` $\{True\}$

    `asegura` $\{sinRepetidos(res) \wedge$
    $(\forall d : \mathbb{Z})((d > 0 \wedge_L n \mod d = 0) \rightarrow_L d \in res) \wedge$
    $(\forall r : \mathbb{Z})(r \in res \rightarrow_L (r > 0 \wedge_L n \mod r = 0))\}$

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

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

    `asegura` {
        $factoriza(n, res) \wedge$
        $primerosElementosSonPrimos(res) \wedge$
        $estaOrdenadaEstrictamentePorPrimerElemento(res)$
    }

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

        $(\sum_0^{|s|-1} (s[i]_0)^{s[i]_1}) = n$

    }

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

        $(\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L esPrimo(s[i]_0))$

    }

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

$(\forall i : \mathbb{Z})(0 \leq i < |s| - 1 \rightarrow_L (s[i]_0 < s[i+1]_0))$

}

Ejercicio 13. Especificar los siguientes problemas sobre secuencias:

a) Dadas dos secuencias s y t, decidir si s está *incluida* en t, es decir, si todos los elementos de s aparecen en t en igual o mayor cantidad

b) Dadas dos secuencias s y t, devolver su *intersección*, es decir, una secuencia con todos los elementos que aparecen en ambas. Si un mismo elemento tiene repetidos, la secuencia retornada debe contener la cantidad mínima de apariciones del elemento en s y en t

c) Dada una secuencia de números enteros, devolver aquel que divida a más elementos de la secuencia. El elemento tiene que pertenecer a la secuencia original. Si existe más de un elemento que cumple esta propiedad, devolver alguno de ellos

d) Dada una secuencia de secuencias de enteros l, devolver una secuencia de l que contenga el máximo valor. Por ejemplo, si $l = \langle\langle 2,3,5\rangle, \langle 8,1\rangle, \langle 2,8,4,3\rangle\rangle$, devolver $\langle 8,1\rangle$ o $\langle 2,8,4,3\rangle$

e) Dada una secuencia l con todos sus elementos distintos, devolver la secuencia de *partes*, es decir, la secuencia de todas las secuencias incluidas en l, cada una con sus elementos en el mismo orden en que aparecen en l

Solución

a) `proc estaIncluida` (in: s,t: $seq\langle\mathbb{Z}\rangle$) : Bool

`requiere` $\{True\}$

`asegura` $\{res = True \leftrightarrow (\forall e : \mathbb{Z})(e \in s \rightarrow \#apariciones(e,s) \leq \#apariciones(e,t))\}$

b) `proc interseccion` (in s,t: $seq\langle\mathbb{Z}\rangle$) : $seq\langle\mathbb{Z}\rangle$

`requiere` $\{True\}$

`asegura` {

$(\forall r : \mathbb{Z})(r \in res \leftrightarrow (r \in s \wedge r \in t)) \wedge$

$(\forall e : \mathbb{Z})(e \in res \rightarrow_L (\#apariciones(e,res) = min(\#apariciones(e,s), \#apariciones(e,t))))$

}

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

`requiere` $\{True\}$

`asegura` $\{res \in s \wedge (\forall e : \mathbb{Z})(e \in s \rightarrow (divisiblesPor(e,s) \leq divisiblesPor(res,s)))\}$

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

$\sum_{0}^{|s|-1}$ `IfThenElse`$(divideA(n, s[i]),\ 1,\ 0)$

}

`pred divideA` (n: $\mathbb{Z}$, m: $\mathbb{Z}$) {

$n \neq 0 \wedge_L m \mod n = 0$

}

d) `proc sequenciaMaxima` (in l: $seq\langle seq\langle\mathbb{Z}\rangle\rangle$) : $seq\langle\mathbb{Z}\rangle$

`requiere` $\{True\}$

`asegura` {
$res \in l \wedge_L$
$(\exists m : \mathbb{Z})(m \in res \wedge esMaximoGeneral(m, l))$
}

`pred esMaximoGeneral` (m : $\mathbb{Z}$, l :$seq\langle seq\langle\mathbb{Z}\rangle\rangle$) {

$(\forall s : seq\langle\mathbb{Z}\rangle)(s \in l \rightarrow esMaximo(m, s))$

}

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

$(\forall n : \mathbb{Z})(n \in s \rightarrow n \leq m)$

}

e) `proc partes` (in l: $seq\langle\mathbb{Z}\rangle$) : $seq\langle seq\langle\mathbb{Z}\rangle\rangle$

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

`asegura` {
$|res| = 2^{|l|} \wedge$
$(\forall i : \mathbb{Z})(0 \leq i < |res|) \rightarrow (estáIncluída(res[i], l) \wedge respetaOrden(res[i], l)) \wedge$
$(\forall i, j : \mathbb{Z})(0 \leq i, j < |res| \wedge i \neq j) \rightarrow (sonDistintas(res[i], res[j]))$
}

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

$(\forall i : \mathbb{Z})(0 \leq i < |s|) \rightarrow (\#apariciones(s[i], s) \leq \#apariciones(s[i], t))$

}

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

$(\forall i, j : \mathbb{Z})((0 \leq i < j < |s|) \rightarrow_L ((\forall n, m : \mathbb{Z})(0 \leq n, m < |l| \rightarrow_L ((s[i] = l[n] \wedge s[j] = l[m]) \leftrightarrow n < m)))$

}

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

$|s| \neq |t| \vee_L (\exists i : \mathbb{Z})(0 \leq i < |s|) \wedge (s[i] \neq t[i])$

}

2.5. Especificación de problemas usando **inout**

Ejercicio 14. Dados dos enteros a y b, se necesita calcular su suma y retornarla en un entero c. ¿Cúales de las siguientes especificaciones son correctas para este problema? Para las que no lo son, indicar por qué.

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

`requiere` $\{True\}$

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

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

`requiere` $\{True\}$

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

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

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

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

Solución

a) Mal, e debería especificar que las variables a y b no cambian de valor agregando $a = old(a) \land b = old(b)$ y se debería hacer referencia a los valores iniciales de las variables a y b utilizando $old(a) + old(b) = c$.

b) Bien, ya que no es necesario restringir la entrada y al hacer referencia a c en la poscondición se predica sobre el valor de salida que debe tener. Otra variante, requeriría tener $C_0 = c$ en la precondición. Ambas son válidas.

c) Bien, es válido utilizar la especificación $L_0 = l$ en la precondición como referir a $old(l)$ en la poscondición. La teórica solamente introduce la notación con *old*.

Ejercicio 15. Dada una secuencia l, se desea sacar su primer elemento y devolverlo. Decidir cúales de estas especificaciones son correctas. Para las que no lo son, indicar por qué y justificar con ejemplos.

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

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

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

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

requiere $\{|l| > 0 \land l = L_0\}$

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

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

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

asegura $\{res = \mathsf{head}(L_0) \land |l| = |L_0| - 1\}$

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

requiere $\{|l| > 0 \land l = L_0\}$

asegura $\{res = \mathsf{head}(L_0) \land l = \mathsf{tail}(L_0)\}$

Solución

a) No define L_0 en la precondición $L_0 = l$.

b) No saca el primero al devolverlo.

c) No especifica qué elemento saca de la secuencia original.

d) *Con voz de locutor de programa de concursos:* ¡Correcto!

Ejercicio 16. Dada una secuencia de enteros, se requiere multiplicar por 2 aquéllos valores que se encuentran en posiciones pares. Indicar por qué son incorrectas las siguientes especificaciones y proponer una alternativa correcta.

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

requiere $\{l = L_0\}$

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

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

requiere $\{l = L_0\}$

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

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

requiere $\{True\}$

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

Solución

a) Se debería especificar que las posiciones impares no se modifican.

b) Se debería especificar que la longitud de la secuencia no se modifica, en caso contrario, se podrían agregar elementos que no estaban originalmente.

c) No especifica si se modifica la secuencia l que es parámetro de entrada salida. Debería especificar que el valor contra el que compara res es el valor original de l, podría ser usando $old(l)$ o usando L_0 e introduciendo $L_0 = l$ en la precondición.

d) Solución propuesta:

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

requiere $\{l = L_0\}$

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

Ejercicio 17. Especificar los siguientes problemas de modificación de secuencias:

a) proc primosHermanos(inout l : $seq\langle\mathbb{Z}\rangle$), que dada una secuencia de enteros mayores a dos, reemplaza dichos valores por el número primo menor más cercano. Por ejemplo, si $l = \langle 6, 5, 9, 14\rangle$, luego de aplicar primosHermanos(l), $l = \langle 5, 3, 7, 13\rangle$

b) proc reemplazar(inout l : $seq\langle\mathsf{Char}\rangle$, in a, b : Char), que reemplaza todas las apariciones de a en l por b

c) proc limpiarDuplicados(inout l : $seq\langle\mathsf{Char}\rangle$) : $seq\langle\mathsf{Char}\rangle$, que elimina los elementos duplicados de l dejando sólo su primera aparición (en el orden original). Devuelve además una secuencia con todas las apariciones eliminadas (en cualquier orden)

Solución

a) proc $\textsf{primosHermanos}$(inout $l: seq\langle\mathbb{Z}\rangle$), que dada una secuencia de enteros mayores a dos, reemplaza dichos valores por el número primo menor más cercano.

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

   `requiere` $\{l = L_0 \land (\forall i : \mathbb{Z})((0 \leq i < |l|) \to (l[i] > 2))\}$

   `asegura` $\{$
   $|l| = |L_0| \land_L$
   $(\forall i : \mathbb{Z})((0 \leq i < |l|) \to esPrimoMasCercano(l[i], L_0[i]))$
   $\}$

   `pred esPrimoMasCercano` (p, n) $\{$

   $esPrimo(p) \land p < n \land (\forall q : \mathbb{Z})((esPrimo(q) \land q < n) \to (q \leq p))$

   $\}$

b) proc $\textsf{reemplazar}$(inout $l : seq\langle\textsf{Char}\rangle$, in $a, b : \textsf{Char}$), que reemplaza todas las apariciones de a en l por b

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

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

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

c) proc $\textsf{limpiarDuplicados}$(inout $l : seq\langle\textsf{Char}\rangle) : seq\langle\textsf{Char}\rangle$, que elimina los elementos duplicados de l dejando sólo su primera aparición (en el orden original). Devuelve además una secuencia con todas las apariciones eliminadas (en cualquier orden)

   `proc limpiarDuplicaods` (inout l: $seq\langle\textsf{Char}\rangle$) : $seq\langle\textsf{Char}\rangle$

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

   `asegura` $\{$
   $|l| \leq |L_0| \land$
   $sinRepetidos(l) \land$
   $respetaOrden(l, L_0) \land$
   $continenMismosElementos(l, L_0) \land$
   $(\forall c : \textsf{Char})(apariciones(c, res) = apariciones(c, L_0) - 1)$
   $\}$

   `pred contienenMismosElementos` (s, t : $seq\langle\textsf{Char}\rangle$) $\{$

   $(\forall e : \mathbb{Z})(e \in s \leftrightarrow e \in t)$

   $\}$

2.6. Ejercicios de parcial

Ejercicio 18. Especificar los siguientes problemas. En todos los casos es recomendable ayudarse escribiendo predicados y funciones auxiliares.

a) Se desea especificar el problema `reemplazarNúmerosPerfectos`, que dada una secuencia de enteros devuelve la secuencia pero con los valores que se corresponden con números perfectos reemplazados por el índice donde se encuentran. Se llama números perfectos a aquellos naturales mayores a cero que son iguales a la suma de sus divisores positivos propios (divisores incluyendo al 1 y sin incluir al propio número). Por ejemplo, $reemplazarNúmerosPerfectos([0, 3, 9, 6, 4, 28, 7]) = [0, 3, 9, 3, 4, 5, 7]$, donde los únicos números reemplazados son el 6 y el 28 porque son los únicos números perfectos de la secuencia.

b) Se desea especificar el problema `ordenarYBuscarMayor` que dada una secuencia s de enteros (que puede tener repetidos) ordena dicha secuencia en orden creciente de valor absoluto y devuelve el valor del máximo elemento. Por ejemplo,

- $ordenarYBuscarMayor([1, 4, 3, 5, 6, 2, 7]) = [1, 2, 3, 4, 5, 6, 7], 7$
- $ordenarYBuscarMayor([1, -2, 2, 5, 1, 4, -2, -10]) = [1, 1, -2, -2, 2, 4, 5, -10], 5$
- $ordenarYBuscarMayor([-10, -3, -7, -9]) = [-3, -7, -9, -10], -3$

c) Se desea especificar el problema *primosEnCero* que dada una secuencia s de enteros devuelve la secuencia pero con los valores que se encuentran en posiciones correspondientes a un número primo reemplazados por 0. Por ejemplo,

- $primosEnCero([0, 1, 2, 3, 4, 5, 6]) = [0, 1, 0, 0, 4, 0, 6]$
- $primosEnCero([5, 7, -2, 13, -9, 1]) = [5, 7, 0, 0, -9, 0]$

d) Se desea especificar el problema *positivosAumentados* que dada una secuencia s de enteros devuelve la secuencia pero con los valores positivos reemplazados por su valor multiplicado por la posición en que se encuentra.

- $positivosAumentados([0, 1, 2, 3, 4, 5]) = [0, 1, 4, 9, 16, 25]$
- $positivosAumentados([-2, -1, 5, 3, 0, -4, 7]) = [-2, -1, 10, 9, 0, -4, 42]$

e) Se desea especificar el problema `procesarPrefijos` que dada una secuencia s de palabras y una palabra p, remueve todas las palabras de s que no tengan como prefijo a p y además retorna la longitud de la palabra más larga que tiene de prefijo a p. Por ejemplo, dados: `s = ["casa", "calamar", "banco", "recuperatorio", "aprobar", "cansado"]` y `p = "ca"` un posible valor para la secuencia s luego de aplicar `procesarPrefijos(s, p)` puede ser `["casa", "calamar", "cansado"]` y el valor devuelto será `7`.

Solución

a) reemplazarNúmerosPerfectos

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

`requiere` $\{True\}$

`asegura` $\{$

$|l| = |res| \wedge_L ($

$perfectosReemplazados(l, res) \wedge$

$noPerfectosIguales(l, res)$

$)$

$\}$

`pred perfectosReemplazados` (l: $seq\langle\mathbb{Z}\rangle$, res: $seq\langle\mathbb{Z}\rangle$) $\{$

$(\forall i : \mathbb{Z})((0 \leq i < |l| \wedge_L esPerfecto(l[i])) \rightarrow_L res[i] = i)$

$\}$

`pred noPerfectosIguales` (l: $seq\langle\mathbb{Z}\rangle$, res: $seq\langle\mathbb{Z}\rangle$) $\{$

$(\forall i : \mathbb{Z})((0 \leq i < |l| \wedge_L \neg esPerfecto(l[i])) \rightarrow_L res[i] = l[i])$

$\}$

`pred esPerfecto` (n : $\mathbb{Z}$) $\{$

$n > 0 \wedge n = \sum_{i=1}^{n-1} \texttt{IfThenElse}(esDivisor(n, i),\ i,\ 0)$

$\}$

`pred esDivisor` (n : $\mathbb{Z}$, i : $\mathbb{Z}$) $\{$

$n \mod i = 0$

$\}$

b) ordenarYBuscarMayor

`proc ordenarYBuscarMayor` (inout s: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$

`requiere` $\{s = S_0\}$

asegura {
$mismosElementos(s, S_0) \wedge_L$
$estaOrdenadaPorValorAbs(s) \wedge$
$esMaximo(s, res)$
}

pred mismosElementos (s: $seq\langle\mathbb{Z}\rangle$, t : $seq\langle\mathbb{Z}\rangle$) {
$(\forall e : \mathbb{Z})(apariciones(s, e) = apariciones(t, e))$
}

pred apariciones (s: $seq\langle\mathbb{Z}\rangle$, e: $\mathbb{Z}$) {
$\sum_{i=0}^{|s|-1} \texttt{IfThenElse}(s[i] = e,\ 1,\ 0)$
}

pred estaOrdenadoPorValorAbs (s: $seq\langle\mathbb{Z}\rangle$) {
$(\forall i : \mathbb{Z})(0 \leq i < |s| - 1 \rightarrow_L |s[i]| \leq |s[i+1]|)$
}

pred esMaximo (s: $seq\langle\mathbb{Z}\rangle$, m: $\mathbb{Z}$) {
$(\exists i : \mathbb{Z})(0 \leq i < |s| \wedge_L s[i] = m) \wedge$
$(\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L s[i] \leq m)$
}

c) primosEnCero

proc primosEnCero (in s: $seq\langle\mathbb{Z}\rangle$) : $seq\langle\mathbb{Z}\rangle$
requiere $\{True\}$
asegura {
$|res| = |s| \wedge_L$ (
$primosReemplazados(s, res) \wedge$
$noPrimosIguales(s, res)$
)
}

pred primosReemplazados (s: $seq\langle\mathbb{Z}\rangle$, res : $seq\langle\mathbb{Z}\rangle$) {
$(\forall i : \mathbb{Z})((0 \leq i < |s| \wedge esPrimo(i)) \rightarrow_L res[i] = 0)$
}

pred noPrimosIguales (s: $seq\langle\mathbb{Z}\rangle$, res : $seq\langle\mathbb{Z}\rangle$) {
$(\forall i : \mathbb{Z})((0 \leq i < |s| \wedge \neg esPrimo(i)) \rightarrow_L res[i] = s[i])$
}

d) positivosAumentados

proc positivosAumentados (in s: $seq\langle\mathbb{Z}\rangle$) : $seq\langle\mathbb{Z}\rangle$
requiere $\{true\}$
asegura {
$|res| = |s| \wedge_L$ (
$positivosCambiados(s, res) \wedge$
$negativosIguales(s, res)$
)
}

pred positivosCambiados (s: $seq\langle\mathbb{Z}\rangle$, res: $seq\langle\mathbb{Z}\rangle$) {
$(\forall i : \mathbb{Z})((0 \leq i < |s| \wedge_L s[i] > 0) \rightarrow_L res[i] = s[i] * i)$
}

pred negativosIguales (s: $seq\langle\mathbb{Z}\rangle$, res: $seq\langle\mathbb{Z}\rangle$) {

$(\forall i : \mathbb{Z})((0 \leq i < |s| \wedge_L s[i] \leq 0) \rightarrow_L res[i] = s[i])$

}

e) procesarPrefijos

`proc` `procesarPrefijos` (inout s: $seq\langle string\rangle$, in p: string) : $\mathbb{Z}$

`requiere` $\{s = S_0\}$

`asegura` {

$(\forall r : string)(r \in s \leftrightarrow (r \in S_0 \wedge \neg esPrefijo(r, p))) \wedge$
maxTamañoConPrefijo(S_0, p, res)

}

`pred` `maxTamañoConPrefijo` (s: $seq\langle string\rangle$, p: string, m: $\mathbb{Z}$) {

$(\exists r : string)(r \in s \wedge esPrefijo(r, p) \wedge |r| = m) \wedge$
$(\forall r : string)(r \in s \wedge esPrefijo(r, p) \rightarrow |r| \leq m)$

}