Algoritmos y Estructuras de Datos

Clase Práctica 2 - Especificación

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

1. Resolución de ejercicios de la clase

Ejercicio 1. Especificar un problema donde dado un número real se quiere obtener su inverso.

`proc inverso` (in n: $\mathbb{R}$) : $\mathbb{R}$
`requiere` $\{n \neq 0\}$
`asegura` $\{result = 1/n\}$

1.1. Especificar los siguientes problemas

Ejercicio 2. Dar el cuadrado del mínimo de una secuencia

`proc cuadradoDelMinimo` (in s: $seq\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$
`requiere` $\{|s| > 0\}$
`asegura` $\{esCuadradoDelMinimo(result, s)\}$

`pred esCuadradoDelMinimo` (e: $\mathbb{Z}$, s: $seq\langle\mathbb{Z}\rangle$) {
$(\exists x : \mathbb{Z})(esMinimo(x, s) \land e = x^2)$
}

`pred esMinimo` (e: $\mathbb{Z}$, s: $seq\langle\mathbb{Z}\rangle$) {
$e \in s \land esMenorOIgualATodos(e, s)$
}
`pred esMenorOIgualATodos` (e: $\mathbb{Z}$, s: $seq\langle\mathbb{Z}\rangle$) {
$(\forall x : \mathbb{Z})(x \in s \rightarrow e \leq x)$
}

Aclaraciones

Notar que no permitimos en la precondición secuencias vacías. Esto se debe a que una secuencia vacía no tiene mínimo porque no tiene elementos, por esa razón esa entrada no tiene ninguna salida posible en nuestro problema.

Ejercicio 3. Dado un entero decidir si es primo

`proc esNumeroPrimo` (in n: $\mathbb{Z}$) : Bool
`requiere` $\{True\}$
`asegura` $\{esPrimo(n) \longleftrightarrow res = true\}$

Aclaraciones

Notar que en $res = true$ estamos utilizando el valor *true* del tipo Bool. No es correcto usar True (fórmula lógica) porque en ese caso estaríamos comparando el valor de un tipo de dato con una fórmula lógica, pertenecen a universos distintos.

Ejercicio 4. Dado un número natural n (mayor que 0), obtener la lista de todos los números naturales que lo dividen

Estrategia para encarar esta especificación

1. Escribir condiciones que quieran que cumpla su especificación en castellano
2. Definir los nombres de los predicados que representan esas condiciones y utilizarlos en la pre y post.
3. Por último, definir esos predicados usando nuestro lenguaje de especificación.

Paso 1

Escribir en castellano condiciones que tiene que cumplir la precondición

1. n debe ser mayor a 0, lo dice explícitamente en el enunciado.

Escribir en castellano condiciones que tiene que cumplir la postcondición

1. Todos los elementos de la lista son naturales y dividen a n
2. Todos los naturales que dividen a n están en la lista (Otra forma de decir lo mismo es: no falta ninguno de los naturales que dividen a n)
3. La lista no tiene repetidos

Paso 2 - Modularizar usando predicados

```
proc divisoresNaturales (in n: ℤ) : seq⟨ℤ⟩
      requiere {n > 0}
      asegura {
            todosSonDivisoresNaturales(n, result) ∧
            noFaltaNingunDivisorNatural(n, result) ∧
            sinRepetidos(result)
      }
```

Paso 3 - Definimos predicados

```
pred todosSonDivisoresNaturales (n: ℤ, s: seq⟨ℤ⟩) {
          (∀e : ℤ)(e ∈ s → esDivisorNatural(e, n))
}
pred noFaltaNingunDivisorNatural (n: ℤ, s: seq⟨ℤ⟩) {
          (∀e : ℤ)(esDivisorNatural(e, n) → e ∈ s)
}
pred esDivisorNatural (e: ℤ, n: ℤ) {
    0 < e ∧_L  n  mód e = 0
}
pred sinRepetidos (s: seq⟨ℤ⟩) {
    Queda de tarea
}
```

Podemos simplificar todosSonDivisoresNaturales y noFaltaNingunDivisorNatural con un si solo si:

```
pred todosSonDivisoresNaturalesYNoFaltaNinguno (n: ℤ, s: seq⟨ℤ⟩) {
          (∀e : ℤ)(e ∈ s ⟷ esDivisorNatural(e, n))
}

proc divisoresNaturales (in n: ℤ) : seq⟨ℤ⟩
      requiere {n > 0}
      asegura {
            todosSonDivisoresNaturalesYNoFaltaNinguno(n, result) ∧
            sinRepetidos(result)
      }
```

Ejercicio 5. Especificar el siguiente problema: "Dado un string de longitud par, reemplazar los espacios blancos por guiones bajos".

Condiciones que tiene que cumplir la precondición

1. El largo del string (secuencia de caracteres) debe ser par (lo dice explícitamente el enunciado)

Condiciones que tiene que cumplir la postcondición

1. El largo de la secuencia sigue siendo el mismo
2. Todas las posiciones de la secuencia donde había espacios blancos ahora hay guiones bajos
3. Todas las posiciones de la secuencia donde no había espacios blancos siguen teniendo el mismo valor (no fueron modificadas)

```
proc reemplazarEspaciosPorGuionesBajos (inout string: seq⟨Char⟩)
      requiere {esPar(|string|)}
      asegura {
            |old(string)| = |string| ∧_L
            (∀i : ℤ)((0 ≤ i < |string| ∧_L old(string)[i] = ' ') →_L string[i] = '_')  ∧
            (∀i : ℤ)((0 ≤ i < |string| ∧_L old(string)[i] ≠ ' ') →_L string[i] = old(string)[i])
      }
```

Aclaraciones

Notar que de no agregar $|old(string)| = |string|$ las fórmulas que le siguen podrían indefirse.

Ejercicio 6. Ejercicio de parcial.

Disclaimer: Recomendamos fuertemente que primero intenten resolverlo por su cuenta y luego vayan a la solución. Obviamente, cualquier cosa que les genere dudas nos consultan.

Ejercicio 3. [40 puntos] Dada una secuencia de enteros s, denominamos corte de s a cualquier par de secuencias de enteros tal que al que concatenar la primera con la segunda se obtiene s. Por ejemplo, los posibles cortes de la secuencia $< 1, 3, 2 >$ son $(<>, < 1, 3, 2 >)$, $(< 1 >, < 3, 2 >)$, $(< 1, 3 >, < 2 >)$ y $(< 1, 3, 2 >, <>)$.

b) [30 puntos] Especificar un problema que dada una secuencia de enteros, indique el corte más parejo posible respecto de la suma de sus elementos. Por ejemplo, dada la secuencia $< 1, 2, 3, 4, 2 >$, el corte más parejo es $(< 1, 2, 3 >, < 4, 2 >)$, porque la diferencia entre la suma de los elementos de cada secuencia es cero.
Notar que puede haber más de un resultado posible y que la diferencia del corte más parejo no siempre es cero: por ejemplo, para la secuencia $< 1, 2, 1 >$ los cortes óptimos son $(< 1 >, < 2, 1 >)$ y $(< 1, 2 >, < 1 >)$.

Utilizamos la estrategia del ejercicio 4

Paso 1

Escribir en castellano condiciones que tiene que cumplir la precondición

No vamos a restringir ninguna posible secuencia de entrada en la precondición. El único caso polémico es, el de siempre, cuando la lista está vacía. Lo que tenemos que preguntarnos es si existe un corte en una lista vacía. Si nos apegamos a la definición del enunciado, "denominamos corte de s a cualquier par de secuencias de enteros que al concatenar la primera con la segunda se obtiene s", podemos ver que que la lista vacía tiene un corte: $(\langle\rangle, \langle\rangle)$.

Por lo tanto, como al tener un corte va a tener un corte más parejo, eso significa que hay una salida posible para esa entrada, haciéndola una entrada que debemos permitir en nuestra precondición.

Escribir en castellano condiciones que tiene que cumplir la postcondición

1. El resultado debe ser un corte de s
2. El resultado debe ser más o igual de parejo que todos los cortes de s

Paso 2 - Modularizar usando predicados

$seq\langle\mathbb{Z}\rangle \times seq\langle\mathbb{Z}\rangle$: Tupla donde el primer y segundo elemento tienen tipo $seq\langle\mathbb{Z}\rangle$

```
proc corteMasParejo (in s: seq⟨ℤ⟩) : seq⟨ℤ⟩ × seq⟨ℤ⟩
      requiere {True}
      asegura {
            esCorte(result, s) ∧
            esMasOIgualDeParejoQueTodosLosCortes(result, s)
      }
```

Paso 3 - Definimos predicados

```
pred esCorte (corte: seq⟨ℤ⟩ × seq⟨ℤ⟩, s: seq⟨ℤ⟩) {
    concat(corte_0, corte_1) = s
}
```

```
pred esMasOIgualDeParejoQueTodosLosCortes (corte: seq⟨ℤ⟩ × seq⟨ℤ⟩, s: seq⟨ℤ⟩) {
    (∀c : seq⟨ℤ⟩ × seq⟨ℤ⟩)(esCorte(c, s) → esMasOIgualDeParejo(corte, c))
}
```

```
pred esMasOIgualDeParejo (corte1: seq⟨ℤ⟩ × seq⟨ℤ⟩, corte2: seq⟨ℤ⟩ × seq⟨ℤ⟩) {
    diferencia(corte1) ≤ diferencia(corte2)
}
```

aux diferencia$(corte : seq\langle\mathbb{Z}\rangle \times seq\langle\mathbb{Z}\rangle) : \mathbb{Z} = abs(total(corte_0) - total(corte_1))$

aux total$(s : seq\langle\mathbb{Z}\rangle) : \mathbb{Z} = \sum_{i=0}^{|s|-1} s[i]$