Lenguaje de especificación

13 de abril de 2018

¿Qué vamos a ver?

- Especificación
- Repaso del lenguaje de especificación
- Ejercicios

- ¿Por qué queremos especificar problemas formalmente?

- ¿Por qué queremos especificar problemas formalmente?
    - Ayuda a entender mejor el problema.

- ¿Por qué queremos especificar problemas formalmente?
  - Ayuda a entender mejor el problema.
  - Para comunicarnos con mayor precisión (porque el lenguaje natural es ambiguo).

- ¿Por qué queremos especificar problemas formalmente?
    - Ayuda a entender mejor el problema.
    - Para comunicarnos con mayor precisión (porque el lenguaje natural es ambiguo).
    - Nos sirve para expresar QUÉ debe cumplir una posible solución de un problema dado.

- ¿Por qué queremos especificar problemas formalmente?
  - Ayuda a entender mejor el problema.
  - Para comunicarnos con mayor precisión (porque el lenguaje natural es ambiguo).
  - Nos sirve para expresar QUÉ debe cumplir una posible solución de un problema dado.
  - No expresamos CÓMO solucionarlo.

Un contrato


desmotivaciones.es
¿Que Es Un Contrato?
El diccionario lo define como
un acuerdo que no se puede romper...
...QUE NO SE PUEDE ROMPER...

```
proc nombre (parametros) {
  Pre {expresionBooleana1}
  Post {expresionBooleana2}
}
```

Sentencias Precondición y Postcondición

Sentencias Precondición y Postcondición

- Ambas son condiciones booleanas.

Sentencias Precondición y Postcondición

- Ambas son condiciones booleanas.
- Tiene que haber un sólo “pre” y un sólo “post” en cada problema de la especificación.

Sentencias Precondición y Postcondición

- Ambas son condiciones booleanas.
- Tiene que haber un sólo "pre" y un sólo "post" en cada problema de la especificación.
- La precondición (el "pre") es una restricción que las variables de entrada deben respetar para garantizar una correcta solución al problema.

Sentencias Precondición y Postcondición

- Ambas son condiciones booleanas.
- Tiene que haber un sólo "pre" y un sólo "post" en cada problema de la especificación.
- La precondición (el "pre") es una restricción que las variables de entrada deben respetar para garantizar una correcta solución al problema.
- La poscondición (el "post") es una condición que debe cumplir el resultado de un algoritmo para respetar la especificación.

- Los parámetros pueden ser de tres tipos

Tipos de parámetros

- Los parámetros pueden ser de tres tipos
- in: parámetros de entrada, son el/los que debe recibir el programa que implemente la especificación para llegar al resultado.

Tipos de parámetros

- Los parámetros pueden ser de tres tipos
- in: parámetros de entrada, son el/los que debe recibir el programa que implemente la especificación para llegar al resultado.
- out: parámetros de salida, son el/los que debe retornar un programa que implemente la especificación.

Tipos de parámetros

- Los parámetros pueden ser de tres tipos
- in: parámetros de entrada, son el/los que debe recibir el programa que implemente la especificación para llegar al resultado.
- out: parámetros de salida, son el/los que debe retornar un programa que implemente la especificación.
- inout: son en simultáneo parámetros de entrada y de salida, se reciben como parámetros de entrada y se modifican para ser retornados como parámetros de salida.

¡Warning! ¡Importante!

- No podemos usar problemas dentro de la especificación de otros problemas

- No podemos usar problemas dentro de la especificación de otros problemas
- Sí podemos usar predicados y funciones auxiliares.

¡Warning! ¡Importante!

- No podemos usar problemas dentro de la especificación de otros problemas
- Sí podemos usar predicados y funciones auxiliares.
- Funciones auxiliares
  - Facilitan la lectura y escritura de las especificaciones.
  - Asignan un nombre a una expresión.
  - Sintaxis: fun f(parámetros) : tipo = e;

¡Warning! ¡Importante!

- No podemos usar problemas dentro de la especificación de otros problemas
- Sí podemos usar predicados y funciones auxiliares.
- Funciones auxiliares
  - Facilitan la lectura y escritura de las especificaciones.
  - Asignan un nombre a una expresión.
  - Sintaxis: fun f(parámetros) : tipo = e;
- Predicados booleanos
  - Son auxiliares que devuelven algo de tipo booleano.
  - Sintaxis: pred p(parámetros) { predicado }

- Los predicados pueden ser intepretados como el conjunto de estados que lo verifican (ejemplo...)

- Los predicados pueden ser intepretados como el conjunto de estados que lo verifican (ejemplo...)
- OJO: las relaciones de fuerza entre predicados desafían el sentido común (por qué?)

- Los predicados pueden ser intepretados como el conjunto de estados que lo verifican (ejemplo...)
- OJO: las relaciones de fuerza entre predicados desafían el sentido común (por qué?)
- Para pensar: dadas dos especificaciones E_1 y E_2, cómo deberían ser las relaciones de fuerza entre las precondiciones y las poscondiciones para que una implementación de E_1 sea válida también para E_2

- Subespecificar es especificar “menos” de lo que pide el enunciado

- Subespecificar es especificar “menos” de lo que pide el enunciado
- Sobreespecificar es especificar “más” de lo que pide el enunciado

Queremos especificar una función que dados 2 parámetros a y b de tipo Int, me devuelva el cociente entre ambos.

Queremos especificar una función que dados 2 parámetros a y b de tipo Int, me devuelva el cociente entre ambos.

`proc cociente` (in a : $\mathbb{Z}$, in b : $\mathbb{Z}$, out res: $\mathbb{Z}$) {
  `Post` $\{res = a\ /\ b\}$
}

Queremos especificar una función que dados 2 parámetros a y b de tipo Int, me devuelva el cociente entre ambos.

`proc cociente` (in a : $\mathbb{Z}$, in b : $\mathbb{Z}$, out res: $\mathbb{Z}$) {
  `Post` $\{res = a \ / \ b\}$
}

- ¿Está bien esto?

Basta de formalidad: Ejemplo

Queremos especificar una función que dados 2 parámetros a y b de tipo Int, me devuelva el cociente entre ambos.

proc cociente (in a : $\mathbb{Z}$, in b : $\mathbb{Z}$, out res: $\mathbb{Z}$) {
  Post $\{res = a \ / \ b\}$
}

- ¿Está bien esto?
- ¿Qué pasa si b = 0?

Basta de formalidad: Ejemplo

Queremos especificar una función que dados 2 parámetros a y b de tipo Int, me devuelva el cociente entre ambos.

proc cociente (in a : $\mathbb{Z}$, in b : $\mathbb{Z}$, out res: $\mathbb{Z}$) {
  Post $\{res = a \ / \ b\}$
}

- ¿Está bien esto?
- ¿Qué pasa si b $=$ 0?
- ¿Debería haber un requiere que lo restrinja?

Queremos especificar una función que dados 2 parámetros a y b de tipo Int, me devuelva el cociente entre ambos.

`proc cociente` (in a : $\mathbb{Z}$, in b : $\mathbb{Z}$, out res: $\mathbb{Z}$) {
  `Post` $\{res = a \;/\; b\}$
}

- ¿Está bien esto?
- ¿Qué pasa si b = 0?
- ¿Debería haber un requiere que lo restrinja?
- Además, la precondición es obligatoria (si no hay precondición entonces es True).

A VER AHORA...

proc cociente (in a : $\mathbb{Z}$, in b : $\mathbb{Z}$, out res: $\mathbb{Z}$) {
  Pre $\{b \neq 0\}$
  Post $\{res = a \ / \ b\}$
}

Ejercicio 1

Dados dos enteros positivos, especificar el problema de decidir si son coprimos.

Ejercicio 2

Dado un número entero positivo, especificar el problema de devolver la cantidad de factores primos que tiene.

Ejercicio 3

Dado un número natural n, obtener la lista de todos los números naturales que lo dividen.

Dado un número natural n, obtener la lista de todos los números naturales que lo dividen.

`proc obtenerDivisores` (in n: $\mathbb{Z}$, out res: $seq\langle\mathbb{Z}\rangle$) $\{$

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

`Post` $\{(\forall d : \mathbb{Z})(d \in res \to (d > 0 \wedge_L n \; mod \; d = 0))\}$

$\}$

Ejercicio 3

Dado un número natural n, obtener la lista de todos los números naturales que lo dividen.

$\texttt{proc obtenerDivisores}$ (in n: $\mathbb{Z}$, out res: $seq\langle\mathbb{Z}\rangle$) $\{$

  $\texttt{Pre}$ $\{n > 0\}$

  $\texttt{Post}$ $\{(\forall d : \mathbb{Z})(d \in res \rightarrow (d > 0 \land_L n \ mod \ d = 0))\}$

$\}$

- ¿Listo?

Dado un número natural n, obtener la lista de todos los números naturales que lo dividen.

proc `obtenerDivisores` (in n: $\mathbb{Z}$, out res: $seq\langle\mathbb{Z}\rangle$) {
  `Pre` $\{n > 0\}$
  `Post` $\{(\forall d : \mathbb{Z})(d \in res \rightarrow (d > 0 \wedge_L n \ mod \ d = 0))\}$
}

- ¿Listo? Casi...

Dado un número natural n, obtener la lista de todos los números naturales que lo dividen.

proc obtenerDivisores (in n: $\mathbb{Z}$, out res: $seq\langle\mathbb{Z}\rangle$) {
  Pre $\{n > 0\}$
  Post $\{(\forall d : \mathbb{Z})(d \in res \rightarrow (d > 0 \wedge_L n \; mod \; d = 0))\}$
}

- ¿Listo? Casi...
- Notar que si $n = 12$, $res = \langle 2, 6\rangle$ satisface la especificación. Es decir, estamos subespecificando.

- ¿Cómo lo arreglamos?

- ¿Cómo lo arreglamos?
  - Todo lo que está, tiene que estar (no hay cosas de más)
  - Todo lo que tiene que estar, está (no hay cosas de menos)

- ¿Cómo lo arreglamos?
  - Todo lo que está, tiene que estar (no hay cosas de más)
  - Todo lo que tiene que estar, está (no hay cosas de menos)

`proc obtenerDivisores` (in n: $\mathbb{Z}$, out res: $seq\langle\mathbb{Z}\rangle$) {
  `Pre` $\{n > 0\}$
  `Post` $\{(\forall d : \mathbb{Z})(d \in res \rightarrow (d > 0 \wedge_L n \ mod \ d = 0)) \wedge (\forall d : \mathbb{Z})((0 < d \leq n \wedge_L n \ mod \ d = 0) \rightarrow d \in res)\}$
}

Ojo con los repetidos! Las secuencias no son conjuntos.

```
proc obtenerDivisores (in n: ℤ, out res: seq⟨ℤ⟩)  {
  Pre {n > 0}
  Post {todosSonDivisores(n, res) ∧ noFaltanDivisores(n, res) ∧
        noHayRepetidos(res)}

  pred todosSonDivisores (n: ℤ, s: seq⟨ℤ⟩)
        {(∀d : ℤ)(d ∈ s → (d > 0 ∧L n mod d = 0))}
  pred noFaltanDivisores (n: ℤ, s: seq⟨ℤ⟩)
        {(∀d : ℤ)((0 < d ≤ n ∧L n mod d = 0) → d ∈ s)}
  pred noHayRepetidos (s: seq⟨ℤ⟩)
        {(∀d : ℤ)((d ∈ s → #apariciones(s, d) = 1)}
}
```

Dada una lista, devolver una lista de listas que contenga todos sus prefijos, en orden decreciente de longitud.