Especificaciones y Contratos

Algoritmos y Estructuras de Datos

Segundo cuatrimestre 2025

¿Qué vamos a ver hoy?

- Repaso Lógica trivaluada
- Especificación de predicados
- Funciones auxiliares
- Especificación de problemas

Contexto general

- ¿Por qué queremos especificar problemas formalmente?

Contexto general

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

Contexto general

- ¿Por qué queremos especificar problemas formalmente?
    - Ayuda a entender mejor el problema.
    - El lenguaje natural es ambiguo.

Contexto general

- ¿Por qué queremos especificar problemas formalmente?
  - Ayuda a entender mejor el problema.
  - El lenguaje natural es ambiguo.
  - A veces es un paso intermedio en demostraciones formales.

Contexto general

- ¿Por qué queremos especificar problemas formalmente?
  - Ayuda a entender mejor el problema.
  - El lenguaje natural es ambiguo.
  - A veces es un paso intermedio en demostraciones formales.
  - etc.

Contexto general

- ¿Por qué queremos especificar problemas formalmente?
  - Ayuda a entender mejor el problema.
  - El lenguaje natural es ambiguo.
  - A veces es un paso intermedio en demostraciones formales.
  - etc.
  - Nos sirve para expresar formalmente QUÉ debe cumplir una posible solución de un problema dado.

Contexto general

- ¿Por qué queremos especificar problemas formalmente?
  - Ayuda a entender mejor el problema.
  - El lenguaje natural es ambiguo.
  - A veces es un paso intermedio en demostraciones formales.
  - etc.
  - Nos sirve para expresar formalmente QUÉ debe cumplir una posible solución de un problema dado.
  - No expresamos CÓMO solucionarlo (puede no haber solución o quizás no sabemos escribirla).

Repaso con ejemplos

Determinar si las siguientes fórmulas se indefinen o no, y en el caso de que no, determinar si es posible su valor. Solo por cuestiones didácticas consideramos al -2 como primo en ESTA DIAPOSITIVA.

Variables	**Fórmula**	**Valor**
$x = 3$	$x + 2 > 4$	$True$
$a = \perp$	$True \vee \perp$	$\perp$
$a = \perp$	$True \vee_L \perp$	$True$
$s = \langle 7, 3, -2, 5 \rangle$	$s[2] > 0$	$False$
$s = \langle 7, 3, -2, 5 \rangle$	$s[4] > 0$	$\perp$
$s = \langle 7, 3, -2, 5 \rangle$	$(\forall i : \mathbb{Z})\ (0 \leq i < 4 \rightarrow_L esPrimo(s[i]))$	$True$
$s = \langle 7, 3, -2, 5 \rangle$	$(\forall i : \mathbb{Z})\ (0 \leq i < 4 \rightarrow esPrimo(s[i]))$	$\perp$
$s = \langle 7, 3, -2, 5 \rangle$	$(\forall i : \mathbb{Z})\ (0 \leq i < 4 \wedge_L esPrimo(s[i]))$	$False$
$s = \langle 7, 3, -2, 5 \rangle$	$(\forall i : \mathbb{Z})\ (0 \leq i < 4 \rightarrow_L s[i] < s[i+1])$	$\perp$
$s : \mathsf{seq}\langle \mathbb{Z} \rangle$	$(\forall i : \mathbb{Z})\ (0 \leq i < \lvert s \rvert \rightarrow_L s[i] < s[i+1])$	$\perp$
$s : \mathsf{seq}\langle \mathbb{Z} \rangle$	$(\forall i : \mathbb{Z})\ (0 \leq i < \lvert s \rvert - 1 \rightarrow_L s[i] < s[i+1])$	Depende

Predicados

- Se utilizan como un remplazo sintáctico
- Siempre evaluan a una condición de verdad: $True$, $False$ o $\perp$
- Se pueden utilizar otros predicados y auxiliares (próximamente) para definirlos

Ejercicio de calentamiento

Queremos especificar un predicado que sea verdadero cuando un entero n divide a otro entero m

Ejercicio de calentamiento

Queremos especificar un predicado que sea verdadero cuando un entero n divide a otro entero m

$$\texttt{pred divide}(n : \mathbb{Z}, m : \mathbb{Z})\ \ \{$$

Ejercicio de calentamiento

Queremos especificar un predicado que sea verdadero cuando un entero n divide a otro entero m

$$\texttt{pred divide}(n : \mathbb{Z}, m : \mathbb{Z}) \ \{ \ n \neq 0 \wedge_L m \mod n = 0 \ \}$$

Más ejercicios

Especificar un predicado que...

- dado un entero n decida si es primo
- dado un entero n y otro m decida si m es el mayor primo que divide a n
- dado un entero e y una secuencia de enteros s decida si e está en la secuencia s
- dadas dos secuencias de caracteres (char) p y s decida si p es prefijo de s
- dada una secuencia de enteros s decida si todos los números primos están en posiciones pares
- dada una secuencia de enteros s decida si tiene un elemento primo que divide al resto de elementos de la secuencia

Auxiliares

- Se utilizan como un remplazo sintáctico
- Si una auxiliar tiene tipo bool en realidad deberían hacer un predicado

Ejercicio de calentamiento

Queremos especificar un auxiliar que sume 10 a un entero x

Ejercicio de calentamiento

Queremos especificar un auxiliar que sume 10 a un entero x

$$\texttt{aux sumaDiez}(x : \mathbb{Z}) : \mathbb{Z} =$$

Ejercicio de calentamiento

Queremos especificar un auxiliar que sume 10 a un entero x

$$\texttt{aux sumaDiez}(x : \mathbb{Z}) : \mathbb{Z} = x + 10$$

Más ejercicios

Especificar un auxiliar que...

- dada una tupla t con dos enteros obtenga el mayor de ellos
- dado un entero x obtenga su dígito menos significativo
- dada una secuencia de enteros s y un entero e calcule la cantidad de apariciones de e en s
- dada una secuencia de enteros s sume los valores de las posiciones pares de s

Problemas

- Cuando especifiquemos problemas tendremos una sola cláusula “requiere” y una sola cláusula “asegura”, a diferencia de lo que se presentó en la teórica
- No está permitido usar procedimientos dentro de otros procedimientos
- Sí podemos usar predicados y auxiliares dentro de procedimientos (todos los que necesitemos)

Ejercicio de calentamiento

Queremos especificar un problema que dados dos enteros n y m decida si n es múltiplo de m

Ejercicio de calentamiento

Queremos especificar un problema que dados dos enteros n y m decida si n es múltiplo de m

$\texttt{proc esMúltiplo}(\mathsf{in}\ n : \mathbb{Z}, \mathsf{in}\ m : \mathbb{Z}) : \mathsf{bool}\ \{$

Ejercicio de calentamiento

Queremos especificar un problema que dados dos enteros n y m decida si n es múltiplo de m

`proc esMúltiplo`(in $n : \mathbb{Z}$, in $m : \mathbb{Z}$) : bool $\{$

`requiere` $\{\ True\ \}$

Ejercicio de calentamiento

Queremos especificar un problema que dados dos enteros n y m decida si n es múltiplo de m

proc esMúltiplo(in $n : \mathbb{Z}$, in $m : \mathbb{Z}$) : bool {
  requiere { $True$ }
  asegura {
    $res = True \leftrightarrow$
    $((m \neq 0 \land_L n \mod m = 0) \lor (m = 0 \land n = 0))$
  }
}

Ejercicio de calentamiento, Otra opción

Queremos especificar un problema que dados dos enteros n y m decida si n es múltiplo de m

Otra opción

$\texttt{proc esMúltiplo}(\mathsf{in}\ n : \mathbb{Z}, \mathsf{in}\ m : \mathbb{Z}) : \mathsf{bool}\ \{$

$\quad \texttt{requiere}\ \{\ True\ \}$

$\quad \texttt{asegura}\ \{$

$\qquad res = True \leftrightarrow$

$\qquad ((\exists k : \mathbb{Z})\ (m * k = n))$

$\quad \}$

$\}$

Sub y sobreespecificación

Decidir si hay subespecificación/sobreespecificación.

Sub y sobreespecificación

Decidir si hay subespecificación/sobreespecificación.

Dado un número entero, devolver su inverso aditivo

proc inverso(in $n : \mathbb{Z}$) : $\mathbb{Z}$ {

  requiere $\{\ True\ \}$

  asegura $\{\ |n| = |res|\ \}$

}

Sub y sobreespecificación

Decidir si hay subespecificación/sobreespecificación.

Dado un número entero, devolver su inverso aditivo

proc inverso(in $n : \mathbb{Z}$) : $\mathbb{Z}$ {
  requiere { $True$ }
  asegura { $|n| = |res|$ }
}

Subespecificación porque el asegura es más débil que lo que pide el problema. Ejemplo: admite el caso $res = n$

Otro caso

Decidir si hay subespecificación/sobreespecificación.

Otro caso

Decidir si hay subespecificación/sobreespecificación.

Dado un número natural, devolver su sucesor

$\texttt{proc sucesor}(\text{in } n : \mathbb{Z}) : \mathbb{Z}\ \{$

$\quad \texttt{requiere}\ \{\ True\ \}$

$\quad \texttt{asegura}\ \{\ n + 1 = res\ \}$

$\}$

Otro caso

Decidir si hay subespecificación/sobreespecificación.

Dado un número natural, devolver su sucesor

$\texttt{proc sucesor}(\text{in } n : \mathbb{Z}) : \mathbb{Z}$ {
  $\texttt{requiere}$ { $True$ }
  $\texttt{asegura}$ { $n + 1 = res$ }
}

Sobreespecificación porque el requiere es más débil que lo que pide el problema. Ejemplo: admite los casos $n < 0$

Ejercicio 12 (de la guía)

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

Ejercicio 12 (de la guía)

- Dados valores de x y n que hacen verdadera la precondición de $\mathtt{p1}$, demostrar que hacen también verdadera la precondición de $\mathtt{p2}$.
- 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 $\mathtt{p2}$. ¿Será también verdadera la postcondición de $\mathtt{p1}$ con este valor de res?
- ¿Podemos concluir que a satisface la especificación de $\mathtt{p1}$?

Ejercicio 13d (también de la guía)

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

Ejercicio 18a (ya saben de dónde)

Dado una secuencia de enteros l, reemplazar los elementos pares de la secuencia por el siguiente

Ejercicio 18a (ya saben de dónde)

Dado una secuencia de enteros l, reemplazar los elementos pares de la secuencia por el siguiente

$\texttt{proc reemplazarParesPorSiguiente}(\textsf{inout } s : \textsf{seq}\langle\mathbb{Z}\rangle)\ \{$

$\quad\texttt{requiere}\ \{\ True\ \}$

$\quad\texttt{asegura}\ \{$

$\quad(\forall i : \mathbb{Z})\ (0 \leq i < |s| \wedge_L divide(2, s[i]) \rightarrow_L setAt(s, i, s[i] + 1))\ \}$

$\}$

Ejercicio 18a (ya saben de dónde)

Dado una secuencia de enteros l, reemplazar los elementos pares de la secuencia por el siguiente

proc reemplazarParesPorSiguiente(inout s : seq$\langle\mathbb{Z}\rangle$) {
  requiere { $True$ }
  asegura {
  $(\forall i : \mathbb{Z})\ (0 \leq i < |s| \wedge_L divide(2, s[i]) \rightarrow_L setAt(s, i, s[i]+1))$ }
}

Pero no se resuelve de esta forma. Recordar que NO ESTAMOS PROGRAMANDO.

¿Una mejor opción?

```
proc reemplazarParesPorSiguiente(inout s : seq⟨ℤ⟩) {
    requiere { s = S_0 }
    asegura {
        (∀i : ℤ) (0 ≤ i < |s| ∧_L divide(2, s[i]) →_L
        s = setAt(s, i, S_0[i] + 1))
    }
}
```

$$\texttt{proc reemplazarParesPorSiguiente}(\textsf{inout } s : \textsf{seq}\langle\mathbb{Z}\rangle)\ \{$$

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

$$\texttt{asegura}\ \{\ (\forall i : \mathbb{Z})\ (0 \leq i < |s| \wedge_L divide(2, s[i]) \rightarrow_L s = setAt(s, i, S_0[i] + 1))\ \}\ \}$$

¿Una mejor opción?

```
proc reemplazarParesPorSiguiente(inout s : seq⟨ℤ⟩) {
    requiere { s = S_0 }
    asegura {
        (∀i : ℤ) (0 ≤ i < |s| ∧_L divide(2, s[i]) →_L
        s = setAt(s, i, S_0[i] + 1))
    }
}
```

$$\texttt{proc reemplazarParesPorSiguiente}(\mathsf{inout}\ s : \mathsf{seq}\langle\mathbb{Z}\rangle)\ \{$$
$$\texttt{requiere}\ \{\ s = S_0\ \}$$
$$\texttt{asegura}\ \{$$
$$(\forall i : \mathbb{Z})\ (0 \le i < |s| \wedge_L divide(2, s[i]) \rightarrow_L s = setAt(s, i, S_0[i] + 1))$$
$$\}$$
$$\}$$

Todavía no.

¿Ahora sí?

$\texttt{proc reemplazarParesPorSiguiente}(\mathsf{inout}\ s : \mathsf{seq}\langle\mathbb{Z}\rangle)\ \{$

$\quad \texttt{requiere}\ \{\ s = S_0\ \}$

$\quad \texttt{asegura}\ \{$

$\qquad |s| = |S_0| \land_L$

$\qquad (\forall i : \mathbb{Z})\ (0 \leq i < |S_0| \land_L divide(2, S_0[i]) \rightarrow_L s[i] = S_0[i] + 1)$

$\quad \}$

$\}$

¿Ahora sí?

$\texttt{proc reemplazarParesPorSiguiente}(\textsf{inout } s : \textsf{seq}\langle\mathbb{Z}\rangle)$ {

$\quad$ $\texttt{requiere}$ $\{\ s = S_0\ \}$

$\quad$ $\texttt{asegura}$ {

$\qquad |s| = |S_0| \land_L$

$\qquad (\forall i : \mathbb{Z})\ (0 \leq i < |S_0| \land_L divide(2, S_0[i]) \rightarrow_L s[i] = S_0[i] + 1)$

$\quad$ }

}

Mucho mejor. ¿Falta algo?

Versión final

```
proc reemplazarParesPorSiguiente(inout s : seq⟨ℤ⟩) {
    requiere { s = S_0 }
    asegura {
        |s| = |S_0| ∧_L
        (∀i : ℤ) (0 ≤ i < |S_0| ∧_L divide(2, S_0[i]) →_L s[i] = S_0[i] + 1)
        ∧
        (∀i : ℤ) (0 ≤ i < |S_0| ∧_L ¬divide(2, S_0[i]) →_L s[i] = S_0[i])
    }
}
```

$$
\begin{array}{l}
\texttt{proc reemplazarParesPorSiguiente}(\textsf{inout } s : \textsf{seq}\langle\mathbb{Z}\rangle)\ \{\\
\quad \texttt{requiere}\ \{\ s = S_0\ \}\\
\quad \texttt{asegura}\ \{\\
\qquad |s| = |S_0| \land_L\\
\qquad (\forall i : \mathbb{Z})\ (0 \leq i < |S_0| \land_L divide(2, S_0[i]) \rightarrow_L s[i] = S_0[i] + 1)\\
\qquad \land\\
\qquad (\forall i : \mathbb{Z})\ (0 \leq i < |S_0| \land_L \neg divide(2, S_0[i]) \rightarrow_L s[i] = S_0[i])\\
\quad \}\\
\}
\end{array}
$$

Ejercicio 19b (adivinen)

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,

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

Ejercicio 19e (adivinen)

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.