Algoritmos y Estructuras de Datos

Especificación y Contratos

Departamento de Computación - FCEyN - UBA

2024

Especificación, algoritmo, programa

1. **Especificación:** descripción del problema a resolver.
   - ¿Qué problema tenemos?
   - Habitualmente, dada en lenguaje formal.
   - Es un contrato que da las propiedades de los datos de entrada y las propiedades de la solución.

2. **Algoritmo:** descripción de la solución escrita para humanos.
   - ¿Cómo resolvemos el problema?

3. **Programa:** descripción de la solución para ser ejecutada en una computadora.
   - También, ¿cómo resolvemos el problema?
   - Pero descripto en un lenguaje de programación.

Especificación de problemas

- Una especificación es un contrato que define qué se debe resolver y qué propiedades debe tener la solución.
  1. Define el qué y no el cómo.
- La especificación de un problema incluye un conjunto de parámetros: datos de entrada cuyos valores serán conocidos recién al ejecutar el programa.
- Además de cumplir un rol “contractual”, la especificación del problema es insumo para las actividades de ...
  1. testing,
  2. verificación formal de corrección,
  3. derivación formal (construir un programa a partir de la especificación).

Parámetros y tipos de datos

- La especificación de un problema incluye un conjunto de parámetros: datos de entrada cuyos valores serán conocidos recién al ejecutar el programa.
- Cada parámetro tiene un tipo de datos.
  - **Tipo de datos:** Conjunto de valores provisto de ciertas operaciones para trabajar con estos valores.
- Ejemplo 1: parámetros de tipo *fecha*
  - valores: ternas de números enteros
  - operaciones: comparación, obtener el año, ...
- Ejemplo 2: parámetros de tipo *dinero*
  - valores: números reales con dos decimales
  - operaciones: suma, resta, ...

Contratos

- Una especificación es un contrato entre el programador de una función y el usuario de esa función.
- **Ejemplo**: calcular la raíz cuadrada de un número real.
- ¿Cómo es la especificación (informalmente, por ahora) de este problema?
- Para hacer el cálculo, el programa debe recibir un número no negativo.
  - Obligación del usuario: no puede proveer números negativos.
  - Derecho del programador: puede suponer que el argumento recibido no es negativo.
- El resultado va a ser la raíz cuadrada del número recibido.
  - Obligación del programador: debe calcular la raíz, siempre y cuando haya recibido un número no negativo
  - Derecho del usuario: puede suponer que el resultado va a ser correcto

Partes de una especificación (contrato)

1. Encabezado
2. Precondición o cláusula “requiere”
   - Condición sobre los argumentos, que el programador da por cierta.
   - Especifica lo que requiere la función para hacer su tarea.
   - Por ejemplo: “el valor de entrada es un real no negativo”
3. Postcondición o cláusula “asegura”
   - Condición sobre el resultado, que debe ser cumplida por el programador siempre y cuando el usuario haya cumplido la precondición.
   - Especifica lo que la función asegura que se va a cumplir después de llamarla (si se cumplía la precondición).
   - Por ejemplo: “la salida es la raíz cuadrada del valor de entrada”

¿Por qué escribir la especificación del problema?

- Nos ayuda a entender mejor el problema
- Nos ayuda a construir el programa
  - Derivación (Automática) de Programas
- Nos ayuda a prevenir errores en el programa
  - Testing
  - Verificación (Automática) de Programas

Lenguaje de especificación

Definición (Especificación) de un problema

proc *nombre*(*parámetros*) : *salida* {
  requiere { P }
  asegura { Q }
}

- P y Q son predicados, denominados la precondición y la postcondición del procedimiento.
- *nombre*: nombre que le damos al problema
  - será resuelto por una función con ese mismo nombre
- *parámetros*: lista de parámetros separada por comas, donde cada parámetro contiene:
  - Tipo de pasaje (entrada, salida o entrada/salida)
  - Nombre del parámetro
  - Tipo de datos del parámetro
- *salida*: Opcional. Tipo de dato de la salida.

Ejemplos

proc *raizCuadrada*(in $x : \mathbb{R}$) : $\mathbb{R}$ {
  requiere $\{x \geq 0\}$
  asegura $\{result * result = x \wedge result \geq 0\}$
}

proc *sumar*(in $x : \mathbb{Z}$, in $y : \mathbb{Z}$) : $\mathbb{Z}$ {
  requiere $\{True\}$
  asegura $\{result = x + y\}$
}

proc *restar*(in $x : \mathbb{Z}$, in $y : \mathbb{Z}$) : $\mathbb{Z}$ {
  requiere $\{True\}$
  asegura $\{result = x - y\}$
}

proc *cualquieramayor*(in $x : \mathbb{Z}$) : $\mathbb{Z}$ {
  requiere $\{True\}$
  asegura $\{result > x\}$

El contrato

- **Contrato:** *El programador escribe un programa P tal que si el usuario suministra datos que hacen verdadera la precondición, entonces P termina en una cantidad finita de pasos retornando un valor que hace verdadera la postcondición.*
- El programa P es correcto para la especificación dada por la precondición y la postcondición exactamente cuando se cumple el contrato.
- Si el usuario no cumple la precondición y P se cuelga o no cumple la poscondición...
  - ¿el usuario tiene derecho a quejarse?
  - ¿Se cumple el contrato?
- Si el usuario cumple la precondición y P se cuelga o no cumple la poscondición...
  - ¿el usuario tiene derecho a quejarse?
  - ¿Se cumple el contrato?

Interpretando una especificación

- proc *raizCuadrada*(in $x : \mathbb{R}$) : $\mathbb{R}$ {
  requiere $\{x \geq 0\}$
  asegura $\{result * result = x \land result \geq 0\}$
  }
- ¿Qué significa esta especificación?
- Se especifica que si el programa `raizCuadrada` se comienza a ejecutar en un estado que cumple $x \geq 0$, entonces el programa **termina** y el estado final cumple $result * result = x$ y $result \geq 0$.

Otro ejemplo

Dados dos enteros **dividendo** y **divisor**, obtener el cociente entero entre ellos.

$$\begin{array}{l}
\text{proc } \mathit{cociente}(\text{in } \mathit{dividendo} : \mathbb{Z}, \text{in } \mathit{divisor} : \mathbb{Z}) : \mathbb{Z} \ \{ \\
\quad \text{requiere } \{\mathit{divisor} > 0\} \\
\quad \text{asegura } \{ \\
\qquad\quad \mathit{result} * \mathit{divisor} \leq \mathit{dividendo} \\
\qquad \wedge\ (\mathit{result} + 1) * \mathit{divisor} > \mathit{dividendo} \\
\quad \} \\
\}
\end{array}$$

Qué sucede si ejecutamos con ...

- dividendo $= 1$ y divisor $= 0$?
- dividendo $= -4$ y divisor $= -2$, y obtenemos result $= 2$?
- dividendo $= -4$ y divisor $= -2$, y obtenemos result $= 0$?
- dividendo $= 4$ y divisor $= -2$, y el programa no termina?

Pasaje de parámetros

in, out, inout

- Parámetros de entrada (**in**): Si se invoca el procedimiento con el argumento **c** para un parámetro de este tipo, entonces se copia el valor **c** antes de iniciar la ejecución
- Parámetros de salida (**out**): Al finalizar la ejecución del procedimiento se copia el valor al parámetro pasado. No se inicializan, y no se puede hablar de estos parámetros en la precondición.
- Parámetros de entrada-salida (**inout**): Es un parámetro que es a la vez de entrada (se copia el valor del argumento al inicio), como de salida (se copia el valor de la variable al argumento). El efecto final es que la ejecución del procedimiento modifica el valor del parámetro.
- Todos los parámetros con atributo **in** (incluso **inout**) están inicializados

Argumentos que se modifican (inout)

Problema: Incrementar en 1 el argumento de entrada.

- Alternativa sin modificar la entrada (usual).

  proc incremento(in $a : \mathbb{Z}$) : $\mathbb{Z}\{$

  requiere $\{\mathit{True}\}$

  asegura $\{\mathit{result} = a + 1\}$

  $\}$

- Alternativa que modifica la entrada: usamos el mismo argumento para la entrada y para la salida.

  proc incremento-modificando(inout $a : \mathbb{Z}$)$\{$

  requiere $\{\mathit{True}\}$

  asegura $\{a = \mathit{old}(a) + 1\}$

  $\}$

- La expresión $\mathit{old}(a)$ representa el valor inicial de la variable a, la usamos en la postcondición para relacionar el valor de salida de a con su valor inicial.

Sobre-especificación

- Consiste en dar una postcondición más restrictiva que lo que se necesita.
- Limita los posibles algoritmos que resuelven el problema, porque impone más condiciones para la salida, o amplía los datos de entrada.
- Ejemplo:
  proc *distinto*(in x : $\mathbb{Z}$) : $\mathbb{Z}$ {
  requiere $\{True\}$
  asegura $\{result = x + 1\}$
  }
- ... en lugar de:
  proc *distinto*(in x : $\mathbb{Z}$) : $\mathbb{Z}${
  requiere$\{True\}$
  asegura $\{result \neq x\}$
  }

Sub-especificación

- Consiste en dar una una precondición más restrictiva que lo realmente necesario, o bien una postcondición más débil que la que se podría dar.
- Deja afuera datos de entrada o ignora condiciones necesarias para la salida (permite soluciones no deseadas).
- Ejemplo:

  proc *distinto*(in $x : \mathbb{Z}$) : $\mathbb{Z}${
    requiere $\{x > 0\}$
    asegura $\{result \neq x\}$
  }

  ... en vez de:

  proc *distinto*(in $x : \mathbb{Z}$) : $\mathbb{Z}${
    requiere $\{True\}$
    asegura $\{result \neq x\}$
  }

Tipos de datos

- Un **tipo de datos** es un conjunto de valores (el conjunto base del tipo) provisto de una serie de operaciones que involucran a esos valores.
- Para hablar de un elemento de un tipo T en nuestro lenguaje, escribimos un término o expresión
  - Variable de tipo T (ejemplos: x, y, z, etc)
  - Constante de tipo T (ejemplos: 1, -1, $\frac{1}{5}$, 'a', etc)
  - Función (operación) aplicada a otros términos (del tipo T o de otro tipo)
- Todos los tipos tienen un elemento distinguido: $\perp$ o Indef

Tipos de datos de nuestro lenguaje de especificación

- Básicos
  - Enteros ($\mathbb{Z}$)
  - Reales ($\mathbb{R}$)
  - Booleanos (Bool)
  - Caracteres (Char)
- Enumerados
- Uplas
- Secuencias
- Conjuntos

Tipo $\mathbb{Z}$ (números enteros)

- Su conjunto base son los números enteros.
- Constantes: $0 \quad ; \quad 1 \quad ; \quad -1 \quad ; \quad 2 \quad ; \quad -2 \quad ; \quad \ldots$
- Operaciones aritméticas:
  - $a + b$ (suma); $a - b$ (resta); abs(a) (valor absoluto)
  - $a * b$ (multiplicación); a div b (división entera);
  - a mod b (resto de dividir a a por b), a^b o pot(a,b) (potencia)
  - a / b (división, da un valor de $\mathbb{R}$)
- Fórmulas que comparan términos de tipo $\mathbb{Z}$:
  - $a < b$ (menor)
  - $a \leq b$ o a `<=` b (menor o igual)
  - $a > b$ (mayor)
  - $a \geq b$ o a `>=` b (mayor o igual)
  - $a = b$ (iguales)
  - $a \neq b$ (distintos)

Tipo $\mathbb{R}$ (números reales)

- Su conjunto base son los números reales.
- Constantes: $0 \quad ; \quad 1 \quad ; \quad -7 \quad ; \quad 81 \quad ; \quad 7{,}4552 \quad ; \quad \pi \ldots$
- Operaciones aritméticas:
  - Suma, resta y producto (pero no div y mod)
  - a/b (división)
  - $\log_b(a)$ (logaritmo)
  - Funciones trigonométricas
- Fórmulas que comparan términos de tipo $\mathbb{R}$:
  - $a < b$ (menor)
  - $a \leq b$ o a `<=` b (menor o igual)
  - $a > b$ (mayor)
  - $a \geq b$ o a `>=` b (mayor o igual)
  - $a = b$ (iguales)
  - $a \neq b$ (distintos)

Tipo Bool (valor de verdad)

- Su conjunto base es $\mathbb{B} = \{\textbf{true}, \textbf{false}\}$.
- Conectivos lógicos: !, &&, $||$, con la semántica bi-valuada estándar.
- Fórmulas que comparan términos de tipo Bool:
  - $a = b$
  - $a \neq b$ (se puese escribir $a \,! = b$)

Tipo Char (caracteres)

- Sus elementos son las letras, dígitos y símbolos.
- Constantes: $`a`, `b`, `c`, \ldots, `z`, \ldots, `A`, `B`, `C`, \ldots, `Z`, \ldots,$ $`0`, `1`, `2`, \ldots, `9`$ (en el orden dado por el estándar ASCII).
- Función ord, que numera los caracteres, con las siguientes propiedades:
  - $\text{ord}(`a`) + 1 = \text{ord}(`b`)$
  - $\text{ord}(`A`) + 1 = \text{ord}(`B`)$
  - $\text{ord}(`1`) + 1 = \text{ord}(`2`)$
- Función char, de modo tal que si c es cualquier char entonces $\text{char}(\text{ord}(c)) = c$.
- Las comparaciones entre caracteres son comparaciones entre sus órdenes, de modo tal que $a < b$ es equivalente a $\text{ord}(a) < \text{ord}(b)$.

Tipos enumerados

- Cantidad finita de elementos.
  Cada uno, denotado por una constante.

  enum *Nombre* { *constantes* }

- *Nombre* (del tipo): tiene que ser nuevo.
- *constantes*: nombres nuevos separados por comas.
- Convención: todos en mayúsculas.
- `ord`(*a*) da la posición del elemento en la definición (empezando de 0).
- Inversa: se usa el nombre del tipo funciona como inversa de `ord`.

Ejemplo de tipo enumerado

Definimos el tipo `Día` así:

```
enum Día {
  LUN, MAR, MIER, JUE, VIE, SAB, DOM
}
```

- Valen:
  - ord(LUN) $=$ 0
  - Día(2) $=$ MIE
  - JUE $<$ VIE

Tipo upla (o tupla)

- Uplas, de dos o más elementos, cada uno de cualquier tipo.
- $T_0 \times T_1 \times \cdots \times T_k$: Tipo de las k-uplas de elementos de tipos T_0, T_1, ... T_k, respectivamente, donde k es fijo.
- Ejemplos:
  - $\mathbb{Z} \times \mathbb{Z}$ son los pares ordenados de enteros.
  - $\mathbb{Z} \times \text{Char} \times \text{Bool}$ son las triplas ordenadas con un entero, luego un carácter y luego un valor booleano.
- *n*ésimo: $(a_0, \ldots, a_k)_m$ es el valor a_m en caso de que $0 \leq m \leq k$. Si no, está indefinido.
- Ejemplos:
  - $(7, 5)_0 = 7$
  - $(\text{`a'}, DOM, 78)_2 = 78$

Funciones y predicados auxiliares

- Asignan un nombre a una expresión.
- Facilitan la lectura y la escritura de especificaciones.
- Modularizan la especificación.

  aux f(argumentos) : *tipo* $= e$;

  - f es el nombre de la función, que puede usarse en el resto de la especificación en lugar de la expresión e.
  - Los argumentos son opcionales y se reemplazan en e cada vez que se usa f.
  - *tipo* es el tipo del resultado de la función (el tipo de e).

  pred p(argumentos)$\{f\}$

  - p es el nombre del predicado, puede usarse en el resto de la especificación en lugar de la formula f.

Ejemplos de funciones auxiliares

- aux $suc(x : \mathbb{Z}) : \mathbb{Z} = x + 1;$
- aux $e() : \mathbb{R} = 2{,}7182;$
- aux $inverso(x : \mathbb{R}) : \mathbb{R} = 1/x;$
- pred $esPar(n : \mathbb{Z})$ { (n mod 2) $= 0$ }

  pred $esImpar(n : \mathbb{Z})$ { $\neg$ (esPar(n)) }
- pred $esFinde(d :$ Día$)\{d =$ SAB $\vee$ $d =$ DOM$\}$
  Otra forma:
  pred $esFinde2(d :$ Día$)\{d >$ VIE$\}$

Expresiones condicionales

Función que elige entre dos elementos del mismo tipo, según una fórmula lógica (guarda)

- si la guarda es verdadera, elige el primero
- si no, elige el segundo

Por ejemplo

- expresión que devuelve el máximo entre dos elementos:

  $\text{aux máx}(a, b : \mathbb{Z}) : \mathbb{Z} = \text{IfThenElseFi}\langle\mathbb{Z}\rangle(a > b, a, b);$

  cuando los argumentos se deducen del contexto, se puede escribir directamente

  $\text{aux máx}(a, b : \mathbb{Z}) : \mathbb{Z} = \text{IfThenElseFi}(a > b, a, b);$ o bien
  $\text{aux máx}(a, b : \mathbb{Z}) : \mathbb{Z} = \text{if } a > b \text{ then } a \text{ else } b \text{ fi};$

- expresión que dado x devuelve $1/x$ si $x \neq 0$ y 0 sino

  $\text{aux unoSobre}(x : \mathbb{R}) : \mathbb{R} = \underbrace{\text{if } x \neq 0 \text{ then } 1/x \text{ else } 0 \text{ fi};}_{\text{no se indefine cuando } x = 0}$

Definir funciones auxiliares versus especificar problemas

Definimos funciones auxiliares

- Expresiones del lenguaje, que se usan dentro de las especificaciones como reemplazos sintácticos. Son de cualquier tipo.
- Dado que es un reemplazo sintáctico, ¡no se permiten definiciones recursivas!

Especificamos problemas

- Condiciones (el contrato) que debería cumplir un algoritmo para ser solución del problema.
- En una especificación dando la precondición y la postcondición con predicados de primer orden.
- No podemos usar otros problemas en la especificación.
  Sí podemos usar predicados y funciones auxiliares ya definidos.

Especificar problemas

- **Ejemplo:** Especificar el problema de retornar el i-ésimo dígito de la representación decimal del número π.
- proc *piesimo*(in $i : \mathbb{Z}$) : $\mathbb{Z}$ $\{$
  requiere $\{i > 0\}$
  asegura $\{result = \lfloor \pi * 10^i \rfloor \bmod 10\}$
  $\}$

Especificar problemas

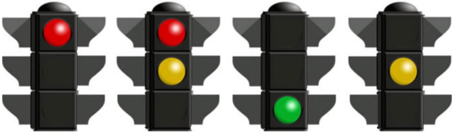
- **Ejemplo:** Especificar un procedimiento que calcule el máximo común divisor (mcd) entre dos números positivos.
- proc $\mathit{mcd}(\text{in } n : \mathbb{Z}, \text{in } m : \mathbb{Z}) : \mathbb{Z}$ {
  requiere $\{n \geq 1 \wedge m \geq 1\}$
  asegura $\{n \bmod \mathit{result} = 0 \wedge m \bmod \mathit{result} = 0 \wedge$
  $\neg(\exists p : \mathbb{Z})(p > \mathit{result} \wedge n \bmod p = 0 \wedge m \bmod p = 0)\}$
  }
- Observar que no damos una fórmula que especifica el valor de retorno, sino que solamente damos las propiedades que debe cumplir!

Especificando un semáforo

- **Ejemplo:** Representamos con tres valores de tipo *Bool* el estado de la luz verde, amarilla y roja de un semáforo.
- Escribir el procedimiento que inicializa el semáforo con la luz roja y el resto de las luces apagadas.
- proc *iniciar*(out v, a, r : Bool) {
  requiere $\{true\}$
  asegura $\{v = \textbf{false} \land a = \textbf{false} \land r = \textbf{true}\}$
  }

Especificando un semáforo

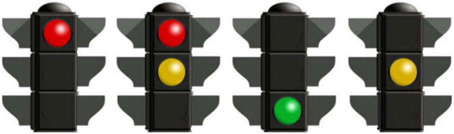
Estado de las luces



- Podemos especificar un predicado para representar cada estado válido del semáforo:
- pred esRojo(v, a, r: Bool) $\{\ v = \textbf{false} \wedge a = \textbf{false} \wedge r = \textbf{true}\ \}$
- pred esRojoAmarillo(v, a, r: Bool) $\{$
  $v = \textbf{false} \wedge a = \textbf{true} \wedge r = \textbf{true}\ \}$
- pred esVerde(v, a, r: Bool)$\{\ v = \textbf{true} \wedge a = \textbf{false} \wedge r = \textbf{false}\ \}$
- pred esAmarillo(v, a, r: Bool)$\{$
  $v = \textbf{false} \wedge a = \textbf{true} \wedge r = \textbf{false}\ \}$

Especificando un semáforo

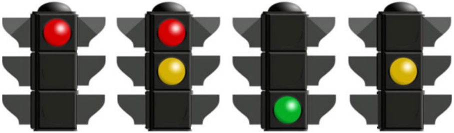
Estado válido



- Podemos especificar un predicado para representar que el semáforo está en un estado válido:
- pred esValido(v, a, r: Bool) {
  $esRojo(v, a, r)$
  $\lor\ esRojoAmarillo(v, a, r)$
  $\lor\ esVerde(v, a, r)$
  $\lor\ esAmarillo(v, a, r)$
  }

Especificando un semáforo

Avance del estado de las luces



- proc *avanzar*(inout v, a, r : Bool) {

  requiere {
  $esValido(v, a, r)$ }

  asegura {
  $(esRojo(old(v), old(a), old(r)) \rightarrow esRojoAmarillo(v, a, r))$
  $\wedge$ $(esRojoAmarillo(old(v), old(a), old(r)) \rightarrow esVerde(v, a, r)$
  $\wedge$ $(esVerde(old(v), old(a), old(r)) \rightarrow esAmarillo(v, a, r))$
  $\wedge$ $(esAmarillo(old(v), old(a), old(r)) \rightarrow esRojo(v, a, r))\}$ }

Volvemos a los Tipos: Secuencias

- **Secuencia:** Varios elementos del mismo tipo T, posiblemente repetidos, ubicados en un cierto orden.
- $seq\langle T\rangle$ es el tipo de las secuencias cuyos elementos son de tipo T.
- T es un tipo arbitrario.
  - Hay secuencias de $\mathbb{Z}$, de Bool, de `Días`, de 5-uplas;
  - también hay secuencias de secuencias de T;
  - etcétera.

Secuencias. Notación

- Una forma de escribir un elemento de tipo $\mathit{seq}\langle T\rangle$ es escribir términos de tipo T separados por comas, entre $\langle \dots \rangle$.
  - $\langle 1, 2, 3, 4, 1, 0\rangle$ es una secuencia de $\mathbb{Z}$.
  - $\langle 1,\ 1+1,\ 3,\ 2*2,\ 5 \bmod 2,\ 0\rangle$ es otra secuencia de $\mathbb{Z}$ (igual a la anterior).
- La secuencia vacía se escribe $\langle\rangle$, cualquiera sea el tipo de los elementos de la secuencia.
- Se puede formar secuencias de elementos de cualquier tipo.
  - Como $\mathit{seq}\langle\mathbb{Z}\rangle$ es un tipo, podemos armar secuencias de $\mathit{seq}\langle\mathbb{Z}\rangle$ (secuencias de secuencias de $\mathbb{Z}$, o sea $\mathit{seq}\langle \mathit{seq}\langle\mathbb{Z}\rangle\rangle$).
  - $\langle\langle 12, 13\rangle, \langle -3, 9, 0\rangle, \langle 5\rangle, \langle\rangle, \langle\rangle, \langle 3\rangle\rangle$ es un elemento de tipo $\mathit{seq}\langle \mathit{seq}\langle\mathbb{Z}\rangle\rangle$.

Secuencias bien formadas

Indicar si las siguientes secuencias están bien formadas. Si están bien formadas, indicar su tipo ($seq\langle\mathbb{Z}\rangle, etc...$)

- $\langle 1, 2, 3, 4, 5\rangle$? Bien Formada. Tipa como $seq\langle\mathbb{Z}\rangle$ y $seq\langle\mathbb{R}\rangle$
- $\langle 1, 2, 3, 4, \frac{1}{0}\rangle$? No está bien formada porque uno de sus componentes está indefinido
- $\langle 1, true, 3, 4, 5\rangle$? No está bien formada porque no es homogénea (Bool y $\mathbb{Z}$)
- $\langle 'a', 2, 3, 4, 5\rangle$? No está bien formada porque no es homogénea ($Char$ y $\mathbb{Z}$)
- $\langle 'H', 'o', 'l', 'a'\rangle$? Bien Formada. Tipa como $seq\langle Char\rangle$
- $\langle true, false, true, true\rangle$? Bien Formada. Tipa como $seq\langle\text{Bool}\rangle$
- $\langle \frac{2}{5}, \pi, e\rangle$? Bien Formada. Tipa como $seq\langle\mathbb{R}\rangle$
- $\langle\rangle$? Bien formada. Tipa como cualquier secuencia $seq\langle X\rangle$ donde X es un tipo válido.
- $\langle\langle\rangle\rangle$? Bien formada. Tipa como cualquier secuencia $seq\langle seq\langle X\rangle\rangle$ donde X es un tipo válido.

Funciones sobre secuencias

Longitud

- $length(a : seq\langle T\rangle) : \mathbb{Z}$
  - Representa la longitud de la secuencia a.
  - Notación: $length(a)$ se puede escribir como $|a|$ o como $a.length$.
- Ejemplos:
  - $|\langle\rangle| = 0$
  - $|\langle 'H', 'o', 'l', 'a'\rangle| = 4$
  - $|\langle 1, 1, 2\rangle| = 3$

Funciones con secuencias

I-ésimo elemento

- Indexación: $seq\langle T\rangle[i:\mathbb{Z}]:T$
  - Requiere $0 \leq i < |a|$.
  - Es el elemento en la i-ésima posición de a.
  - La primera posición es la 0.
  - Notación: $a[i]$.
  - Si no vale $0 \leq i < |a|$ se indefine.
- Ejemplos:
  - $\langle 'H', 'o', 'l', 'a' \rangle[0] = 'H'$
  - $\langle 'H', 'o', 'l', 'a' \rangle[1] = 'o'$
  - $\langle 'H', 'o', 'l', 'a' \rangle[2] = 'l'$
  - $\langle 'H', 'o', 'l', 'a' \rangle[3] = 'a'$
  - $\langle 1, 1, 1, 1\rangle[0] = 1$
  - $\langle\rangle[0] = \perp$ (Indefinido)
  - $\langle 1, 1, 1, 1\rangle[7] = \perp$ (Indefinido)

Funciones con secuencias
Igualdad

Dos secuencias s_0 y s_1 (notación $s_0 = s_1$) son iguales si y sólo si

- Tienen la misma cantidad de elementos
- Dada una posición, el elemento contenido en la secuencia s_0 es igual al elemento contenido en la secuencia s_1.

Ejemplos:

- $\langle 1,2,3,4 \rangle = \langle 1,2,3,4 \rangle$? Sí
- $\langle \rangle = \langle \rangle$? Sí
- $\langle 4,4,4 \rangle = \langle 4,4,4 \rangle$? Sí
- $\langle 1,2,3,4,5 \rangle = \langle 1,2,3,4 \rangle$? No
- $\langle 1,2,3,4,5 \rangle = \langle 1,2,4,5,6 \rangle$? No
- $\langle 1,2,3,5,4 \rangle = \langle 1,2,3,4,5 \rangle$? No

Funciones con secuencias

Cabeza o Head

- Cabeza: $head(a : seq\langle T\rangle) : T$
  - Requiere $|a| > 0$.
  - Es el primer elemento de la secuencia a.
  - Es equivalente a la expresión $a[0]$.
  - Si no vale $|a| > 0$ se indefine.
- Ejemplos:
  - $head(\langle 'H', 'o', 'l', 'a'\rangle) = 'H'$
  - $head(\langle 1, 1, 1, 1\rangle) = 1$
  - $head(\langle\rangle) = \perp$ (Indefinido)

Funciones con secuencias

Cola o Tail

- Cola: $tail(a : seq\langle T\rangle) : seq\langle T\rangle$
  - Es la secuencia resultante de eliminar su primer elemento.
  - Está definida cuando $|a| > 0$. Si no vale esa condición, se indefine.
- Ejemplos:
  - $tail(\langle 'H', 'o', 'l', 'a' \rangle) = \langle 'o', 'l', 'a' \rangle$
  - $tail(\langle 1, 1, 1, 1 \rangle) = \langle 1, 1, 1 \rangle$
  - $tail(\langle\rangle) = \perp$ (Indefinido)
  - $tail(\langle 6 \rangle) = \langle\rangle$

Funciones con secuencias

Agregar al principio o addFirst

- Agregar cabeza: $addFirst(t : T, a : seq\langle T\rangle) : seq\langle T\rangle$
  - Es una secuencia con los elementos de a, agregándole t como primer elemento.
  - Es una función que no se indefine
- Ejemplos:
  - $addFirst('x', \langle 'H', 'o', 'l', 'a'\rangle) = \langle 'x', 'H', 'o', 'l', 'a'\rangle$
  - $addFirst(5, \langle 1, 1, 1, 1\rangle) = \langle 5, 1, 1, 1, 1\rangle$
  - $addFirst(1, \langle\rangle) = \langle 1\rangle$

Funciones con secuencias

Concatenación o concat

- Concatenación: $concat(a : seq\langle T\rangle, b : seq\langle T\rangle) : seq\langle T\rangle$
  - Es una secuencia con los elementos de a, seguidos de los de b.
  - Notación: $concat(a, b)$ se puede escribir $a ++ b$.
- Ejemplos:
  - $concat(\langle'H','o'\rangle, \langle'l','a'\rangle) = \langle'H','o','l','a'\rangle$
  - $concat(\langle 1, 2\rangle, \langle 3, 4\rangle) = \langle 1, 2, 3, 4\rangle$
  - $concat(\langle\rangle, \langle\rangle) = \langle\rangle$
  - $concat(\langle 2, 3\rangle, \langle\rangle) = \langle 2, 3\rangle$
  - $concat(\langle\rangle, \langle 5, 7\rangle) = \langle 5, 7\rangle$

Funciones con secuencias

Subsecuencia o subseq

- Subsecuencia: $subseq(a : seq\langle T\rangle, d, h : \mathbb{Z}) : seq\langle T\rangle$
  - Es una sublista de a en las posiciones entre d (inclusive) y h (exclusive).
  - Cuando $0 \leq d = h \leq |a|$, retorna la secuencia vacía.
  - Cuando no se cumple $0 \leq d \leq h \leq |a|$, se indefine!
- Ejemplos:
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 1) = \langle 'H'\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 4) = \langle 'H', 'o', 'l', 'a'\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 2, 2) = \langle\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, -1, 3) = \perp$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 10) = \perp$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 3, 1) = \perp$

Funciones con secuencias

- Cambiar una posición:
  $setAt(a : seq\langle T\rangle, i : \mathbb{Z}, val : T) : seq\langle T\rangle$
  - Requiere $0 \leq i < |a|$
  - Es una secuencia igual a a, pero con valor val en la posición i.
- Ejemplos:
  - $setAt(\langle 'H', 'o', 'l', 'a'\rangle, 0, 'X') = \langle 'X', 'o', 'l', 'a'\rangle$
  - $setAt(\langle 'H', 'o', 'l', 'a'\rangle, 3, 'A') = \langle 'H', 'o', 'l', 'A'\rangle$
  - $setAt(\langle\rangle, 0, 5) = \perp$ (Indefinido)

Operaciones sobre secuencias - Resumen

- $length(a : seq\langle T\rangle) : \mathbb{Z}$ (notación $|a|$)
- indexación: $seq\langle T\rangle[i : \mathbb{Z}] : T$
- igualdad: $seq\langle T\rangle = seq\langle T\rangle$
- $head(a : seq\langle T\rangle) : T$
- $tail(a : seq\langle T\rangle) : seq\langle T\rangle$
- $addFirst(t : T, a : seq\langle T\rangle) : seq\langle T\rangle$
- $concat(a : seq\langle T\rangle, b : seq\langle T\rangle) : seq\langle T\rangle$ (notación a++b)
- $subseq(a : seq\langle T\rangle, d, h : \mathbb{Z}) : \langle T\rangle$
- $setAt(a : seq\langle T\rangle, i : \mathbb{Z}, val : T) : seq\langle T\rangle$

Lemas sobre secuencias

Sea s_0, s_1 secuencias de tipo T y e un elemento de tipo T. Justificar brevemente por qué cada una de las siguientes afirmaciones son verdaderas:

- $|addFirst(e, s_0)| = 1 + |s_0|$? Sí
- $|concat(s_0, s_1)| = |s_0| + |s_1|$? Sí
- $s_0 = tail(addFirst(e, s_0))$? Sí
- $s_0 = subseq(s_0, 0, |s_0|)$? Sí
- $s_0 = subseq(concat(s_0, s_1), 0, |s_0|)$? Sí
- $head(addFirst(e, s_0)) = e$? Sí
- $addFirst(e, s_0)[0] = e$? Sí
- $addFirst(e, s_0)[0] = head(addFirst(e, s_0))$? Sí

Si hay tiempo: Predicando sobre secuencias

Predicando sobre secuencias

- Crear un predicado que sea **Verdadero** si y sólo si una secuencia de enteros sólo posee enteros mayores a 5.
- Solución:

$$\texttt{pred seq\_gt\_five(s: } \mathit{seq}\langle\mathbb{Z}\rangle\texttt{) \{}$$
$$(\forall i : \mathbb{Z})(0 \leq i < |s| \rightarrow_L s[i] > 5)$$
$$\texttt{\}}$$

Predicando sobre secuencias

- Crear un predicado que sea **Verdadero** si y sólo si todos los elementos con indices pares de una secuencia de enteros s son mayores a 5.

- Solución:

$$
\begin{array}{l}
\texttt{pred seq\_even\_gt\_five(s: } seq\langle\mathbb{Z}\rangle\texttt{) \{} \\
\quad (\forall i : \mathbb{Z})(\\
\qquad ((0 \leq i < |s|) \wedge (i \bmod 2 = 0)) \\
\qquad\quad \rightarrow_L s[i] > 5) \\
\texttt{\}}
\end{array}
$$

Predicando sobre secuencias

- Crear un predicado que sea **Verdadero** si y sólo si hay algún elemento en la secuencia s que sea par y mayor que 5.

- Solución:

  $\texttt{pred seq\_has\_elem\_even\_gt\_five(s: } seq\langle\mathbb{Z}\rangle\texttt{) } \{$
  $\quad (\exists i : \mathbb{Z})($
  $\quad\quad (0 \leq i < |s| \wedge_L ((s[i] \bmod 2 = 0) \wedge (s[i] > 5))$
  $\}$

Predicando sobre secuencias

Secuencia vacía o "isEmpty"

- Definir un predicado `isEmpty` que indique si la secuencia s no tiene elementos.

- Solución

```
pred isEmpty(s: seq⟨T⟩) {
  |s| = 0
}
```

Predicando sobre secuencias

Pertenencia o "has"

- Definir un predicado `has` que indique si el elemento e aparece (al menos una vez) en la secuencia s.
- Solución

  pred has(s: $seq\langle T\rangle$, e: T) {
  $(\exists i : \mathbb{Z})(0 \leq i < |s| \wedge_L s[i] = e)$
  }
- Notación: Podemos utilizar este predicado como $e \in s$

- Definir un predicado `equals(s1,s2)` que indique si la secuencia $s1$ es igual a la secuencia $s2$.

- Solución

```
pred equals(s1, s2: seq⟨T⟩) {
  s1 = s2
}
```

Predicando sobre secuencias

Cambiar un elemento o "setAt"

- Definir un predicado `isSetAt(s1,s2,e,i)` que indique si la secuencia $s2$ cambiandole el elemento de la posición i por e es igual a $s1$.
- En el caso que **no se cumpla** que $0 \leq i < |s2|$, retornar **Falso** sólo si ambas secuencias **no son** iguales.
- Solución

$$
\begin{array}{l}
\texttt{pred isSetAt(s1, s2: } \mathit{seq}\langle T\rangle\texttt{, e: } T\texttt{, i: } \mathbb{Z}\texttt{) \{} \\
\quad (0 \leq i < |s2| \rightarrow_L s1 = \mathit{setAt}(s2, e, i)) \\
\quad \wedge \\
\quad (\neg(0 \leq i < |s2|) \rightarrow s1 = s2) \\
\quad) \\
\texttt{\}}
\end{array}
$$

$\sum$ - Sumatoria

El lenguaje de especificación provee formas de acumular resultados para los tipos numéricos $\mathbb{Z}$ y $\mathbb{R}$.
El término

$$\sum_{i=from}^{to} Expr(i)$$

retorna la suma de todas las expresiones $Expr(i)$ entre *from* y *to*. Es decir,

$$Expr(from) + Expr(from+1) + \cdots + Expr(to-1) + Expr(to)$$

Algunas condiciones:

- $Expr(i)$ debe ser un tipo numérico ($\mathbb{R}$ o $\mathbb{Z}$).
- $from \leq to$ (retorna 0 si no se cumple).
- *from* y *to* es un **rango** (finito) de valores enteros, caso contrario se indefine.
- Si existe i tal que $from \leq i \leq to$ y $Expr(i) = \perp$, entonces toda la expresión se indefine!

$\sum$ - Ejemplos

Retornar la sumatoria de una secuencia s de tipo $seq\langle T\rangle$.
Solución:

$$\sum_{i=0}^{|s|-1} s[i]$$

Ejemplos:

- Si $s = \langle 1, 1, 3, 3\rangle$ retornará

$$s[0] + s[1] + s[2] + s[3] = 1 + 1 + 3 + 3 = 8$$

- Si $s = \langle\rangle$, entonces $from = 0$ y $to = -1$, por lo tanto retornará 0

$\sum$ - Ejemplos

Retornar la sumatoria de la posición 1 (únicamente) de la secuencia s.
Solución:

$$\sum_{i=1}^{1} s[i]$$

Ejemplos:

- Si $s = \langle 7, 11, 3, 3, 2, 4 \rangle$ retornará $s[1] = 11$.
- Si $s = \langle 7 \rangle$ la sumatoria se indefine ya que $s[1] = \perp$.

$\sum$ - Ejemplos

Retornar la sumatoria de los índices pares de la secuencia s.
Solución:

$$\sum_{i=0}^{|s|-1} (\text{if } (i \bmod 2 = 0) \text{ then } s[i] \text{ else } 0 \text{ fi})$$

Ejemplos:

- Si $s = \langle 7, 1, 3, 3, 2, 4 \rangle$ retornará

$$s[0] + 0 + s[2] + 0 + s[4] + 0 = 7 + 0 + 3 + 0 + 2 + 0 = 12$$

- Si $s = \langle 7 \rangle$ retornará $s[0] = 7$.

$\sum$ - Ejemplos

Retornar la sumatoria de los elementos mayores a 0 de la secuencia s.

Solución:

$$\sum_{i=0}^{|s|-1} (\text{if } (s[i] > 0) \text{ then } s[i] \text{ else } 0 \text{ fi})$$

Ejemplos:

- Si $s = \langle 7, 1, -3, 3, 2, -4 \rangle$ retornará

  $s[0] + s[1] + 0 + s[3] + s[4] + 0 = 7 + 1 + 0 + 3 + 2 + 0 = 13$

- Si $s = \langle -7 \rangle$ retornará 0.

$\prod$ - Productoria

El término

$$\prod_{i=from}^{to} Expr(i)$$

retorna el producto de todas las expresiones $Expr(i)$ entre *from* y *to*. Es decir,

$$Expr(from) * Expr(from + 1) * \cdots * Expr(to - 1) * Expr(to)$$

- $Expr(i)$ debe ser un tipo numérico ($\mathbb{R}$ o $\mathbb{Z}$).
- *from* y *to* define un rango de valores enteros (finito) y $from \leq to$ (retorna 1 si no se cumple).
- Si $Expr(i) = \perp$, toda la productoria se indefine.

$\prod$ - Ejemplos

Retornar la productoria de los elementos mayores a 0 de la secuencia s.

Solución:

$$\prod_{i=0}^{|s|-1} (\text{if } (s[i] > 0) \text{ then } s[i] \text{ else } 1 \text{ fi})$$

Ejemplos:

- Si $s = \langle 7, 1, -3, 3, 2, -4 \rangle$ retornará

$$s[0] * s[1] * 1 * s[3] * s[4] * 1 = 7 * 1 * 1 * 3 * 2 * 1 = 42$$

- Si $s = \langle -7 \rangle$ retornará 1.

Algunas funciones auxiliares interesantes

Definir una función que permita contar la cantidad de apariciones de un elemento e en la secuencia s:

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

Ejemplos:

- #apariciones($\langle 5, 1, 1, 1, 3, 3\rangle$, 1)=3
- #apariciones($\langle 5, 1, 1, 1, 3, 3\rangle$, 2)=0
- #apariciones($\langle 5, 1, 1, 1, 3, 3\rangle$, 3)=2
- #apariciones($\langle 5, 1, 1, 1, 3, 3\rangle$, 5)=1
- #apariciones($\langle\rangle$, 5)=0

Algunas funciones auxiliares interesantes

Definir un predicado que sea verdadero si y sólo si una secuencia es una permutación[1] de otra secuencia:

$$\text{pred } \mathit{es\_permutacion}(s1, s2 : seq\langle T\rangle)\{$$
$$(\forall e : T)(\#apariciones(s1, e) = \#apariciones(s2, e))$$
$$\}$$

[1] mismos elementos y misma cantidad por cada elemento, en un orden potencialmente distinto

Un ejemplo con cantidades

Otra forma de definir un predicado que sea verdadero si un número entero n es primo:

$$
\begin{array}{l}
\text{pred } \mathit{soy\_primo}(n : \mathbb{Z})\{ \\
\quad n > 1 \land \\
\quad (\sum_{i=2}^{n-1} (\text{if } (n \bmod i = 0) \text{ then } 1 \text{ else } 0 \text{ fi})) = 0 \\
\}
\end{array}
$$

1. Por cada número entre 2 y $n - 1$ me fijo si n es divisible por ese número.
2. Cada vez que encuentro un número i que me divide, acumulo 1
3. Si al final no acumulé nada, quiere decir que no encontré ningún número entre 2 y $n - 1$ que divida a n

Otro ejemplo con cantidades

Definir una función que retorne la cantidad de números primos menores a un entero n (o 0 si $n < 0$)

aux $\#primosMenores(n : \mathbb{Z}) =$
$\sum_{i=2}^{n-1}$(if $soy\_primo(i)$ then 1 else 0 fi);

1. Por cada número entre 2 y $n - 1$ me fijo si n es primo.
2. Cada vez que encuentro un número primo, acumulo 1
3. Si $n < 0$, entonces $\neg(2 \leq -1)$, por lo que $\sum$ retorna 0.

Contando elementos en un conjunto

- La siguiente expresión es muy común en especificaciones de problemas:
$$\sum_{i \in A} \text{if } P(i) \text{ then } 1 \text{ else } 0 \text{ fi}.$$
- Introducimos la siguiente notación como reemplazo sintáctico para esta sumatoria:
$$\#\{i \in A : P(i)\}$$
- Por ejemplo, podemos escribir
$$\#\{i : 1 \leq i \leq n-1 \ \wedge \ soy\_primo(i)\}.$$
- Observación: A tiene que se un conjunto **finito**.

Sumatoria de secuencias de $\mathbb{R}$

Definir una función que sume los inversos multiplicativos de una lista de reales.
Si no existe el inverso multiplicativo, ignorar el término.

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

Ejemplo de especificación con sumatorias

Especificar un programa que sume los inversos multiplicativos de una lista de reales, pero que requiera que todos los elementos de la secuencia **tengan** inverso multiplicativo.

pred todos_tienen_inverso(s: $seq\langle\mathbb{R}\rangle$) $\{$

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

$\}$

proc sumaInversos(in s: $seq\langle\mathbb{R}\rangle$): $\mathbb{R}$ $\{$

requiere $\{$ todos_tienen_inverso(s) $\}$

asegura $\{$ result= sumarInvertidos(s) $\}$

$\}$

Especificaciones y comentarios

- Los nombres de los predicados/funciones ayudan a describir el significado de las precondiciones y postcondiciones de las especificaciones.
- Los comentarios (/*...*/) también ayudan a describir el significado de las precondiciones y postcondiciones de las especificaciones y son útiles si no hay predicados

Ejemplo:

proc menorElemDistintos(in s: $seq\langle\mathbb{Z}\rangle$): $\mathbb{Z}$ {

requiere { noNegativos(s) $\wedge$ noHayRepetidos(s) }

asegura {

/* result es un indice válido de s */

$0 \leq result < |s| \wedge_L$

/* s[result] es el menor elemento de s */

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

}

Conjuntos

- **Conjuntos** Varios elementos del mismo tipo T, sin repetidos, no importa el orden.
- $conj\langle T\rangle$ es el tipo de los conjuntos cuyos elementos son de tipo T.
- T es un tipo arbitrario.
  - Hay conjuntos de $\mathbb{Z}$, de Bool, de `Días`, de 5-uplas;
  - también hay conjuntos de conjuntos de T;
  - también hay conjuntos de secuencias de T;
  - etcétera.

Conjuntos . Notación

- Usamos la notación matématica clasica. $conj\langle T\rangle$ es escribir términos de tipo T separados por comas, entre $\{\ldots\}$. Los distinguimos de secuencias por el uso de llaves en vez de corchetes.
    - $\{1, 2, 3, 4\}$ es un conjunto de $\mathbb{Z}$.
    - $\{1,\ 1+1,\ 3,\ 2*2,\ ,\ 0\}$ es otro conjunto de $\mathbb{Z}$ (igual a la anterior).
- El conjunto vacío se escribe $\{\}$, cualquiera sea el tipo de los elementos de la secuencia.
- Se puede formar conjuntos de elementos de cualquier tipo, de la misma manera que lo podíamos hacer con las secuencias.

Funciones sobre conjuntos

Cardinal

- $cardinal(a : conj\langle T\rangle) : \mathbb{Z}$
  - Representa el tamaño del conjunto a.
  - Notación: $cardinal(a)$ se puede escribir como $|a|$ o como $a.cardinal$.
- Ejemplos:
  - $|\{\}| = 0$
  - $|\{'H', 'o', 'l', 'a'\}| = 4$
  - $|\{1, 7\}| = 2$

Funciones con conjuntos

Pertenece

- $in(T, conj\langle T\rangle)$: Bool
  - Dice si un elemento se encuentra en el conjunto c.
  - Notación: $e \in c$.
- Ejemplos:
  - $'o' \in \{'H', 'o', 'l', 'a'\} = \texttt{True}$

Funciones con conjuntos

Union

- $union(conj\langle T\rangle, conj\langle T\rangle) : conj\langle T\rangle$
  - Unión de conjuntos.
  - Notación: $c_o \cup c_1$.
- Ejemplos:
  - $\{1, 2\} \cup \{1, 3, 4\} = \{1, 2, 3, 4\}$

Funciones con conjuntos

Intersección

- $intersection(conj\langle T\rangle, conj\langle T\rangle) : conj\langle T\rangle$
  - Intersección de conjuntos.
  - Notación: $c_o \cap c_1$.
- Ejemplo:
  - $\{1,2\} \cap \{1,3,4\} = \{1\}$

Funciones con conjuntos

Diferencia

- $\textit{diff}(\textit{conj}\langle T\rangle, \textit{conj}\langle T\rangle) : \textit{conj}\langle T\rangle$
  - Diferencia: $\{x \in c_0 \wedge x \notin c_1\}$
  - Notación: $c_o - c_1$.
- Ejemplo:
  - $\{1, 2\} - \{1, 3, 4\} = \{2\}$
- Está también la diferencia simétrica, recuerden la definición

Funciones con conjuntos

Igualdad

Dos conjuntos c_0 y c_1 (notación $c_0 = c_1$) son iguales si y sólo si

- Tienen la misma cantidad de elementos
- Tienen los mismos elementos sin importar el orden. Todo elemento x en c_0 deber estar en c_1 y viceversa.

Ejemplos:

- $\{1, 2, 3, 4\} = \{1, 2, 3, 4\}$? Sí
- $\{\} = \{\}$? Sí
- $\{4, 5\} = \{5, 4\}$? Sí
- $\{1, 2, 3, 4, 5\} = \{1, 2, 3, 4\}$? No
- $\{1, 2, 3, 4, 5\} = \{1, 2, 4, 5, 6\}$? No

Si hay tiempo: Matrices

Matrices

- Una matriz es una secuencia de secuencias, todas con la misma longitud (y no ser vacías).
- Cada posición de esta secuencia es a su vez una secuencia, que representa una fila de la matriz.
- Definimos $Mat\langle\mathbb{Z}\rangle$ como un reemplazo sintáctico para $Seq\langle Seq\langle\mathbb{Z}\rangle\rangle$.
- Una $Seq\langle Seq\langle\mathbb{Z}\rangle\rangle$ representa una matriz si todas las secuencias tienen la misma longitud! Definimos entonces:

  pred esMatriz(m: $Seq\langle Seq\langle\mathbb{Z}\rangle\rangle$) {
  $(\forall i : \mathbb{Z})(0 \leq i < filas(m) \rightarrow_L$
  $|m[i]| > 0 \wedge$
  $(\forall j : \mathbb{Z})(0 \leq j < filas(m) \rightarrow_L |m[i]| = |m[j]|))$
  }

- Notar que podemos reemplazar $Mat\langle\mathbb{Z}\rangle$ por $Seq\langle Seq\langle\mathbb{Z}\rangle\rangle$ en la definición del predicado.

Matrices

- Tenemos funciones para obtener la cantidad de filas y columnas de una matriz:

  aux filas($m : Mat\langle\mathbb{Z}\rangle$) : $\mathbb{Z} = |m|$;

  aux columnas($m : Mat\langle\mathbb{Z}\rangle$) : $\mathbb{Z}$
  $= \text{if } filas(m) > 0 \text{ then } |m[0]| \text{ else } 0 \text{ fi}$;

- En muchas ocasiones debemos recibir matrices cuadradas. Definimos también:

  pred esMatrizCuadrada(m: $Seq\langle Seq\langle\mathbb{Z}\rangle\rangle$) {
  $esMatriz(m) \ \land \ filas(m) = columnas(m)$
  }

Matrices

- **Ejemplo:** Un predicado que determina si una matriz es una matriz identidad.

$$
\begin{array}{l}
\text{pred esMatrizIdentidad(m: } Mat\langle\mathbb{Z}\rangle\text{) } \{ \\
\quad esMatrizCuadrada(m) \wedge_L \\
\quad (\\
\qquad (\forall i : \mathbb{Z})\ (0 \leq i < filas(m) \rightarrow_L m[i][i] = 1) \wedge \\
\qquad (\forall i : \mathbb{Z})\ (\forall j : \mathbb{Z})\ (0 \leq i, j < filas(m) \wedge i \neq j \\
\qquad\qquad\qquad\qquad \rightarrow_L m[i][j] = 0) \\
\quad) \\
\}
\end{array}
$$

Bibliografía

- David Gries - The Science of Programming
  - Chapter 4 - Predicates (cuantificación, variables libres y ligadas, etc.)
  - Chapter 5 - Notations and Conventions for Arrays (secuencias)