Universidad Nacional de Rosario

Documentación de lenguaje de especificación formal simplificado

---

Autor: Stizza, Federico

Director: Cristiá, Maximiliano

Departamento de Ciencias de la Computación

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Av. Pellegrini 250, Rosario, Santa Fe, Argentina

13 de Junio de 2023

Índice

1. Introducción

Los **métodos formales** son un conjunto de lenguajes, técnicas rigurosas y herramientas basados en matemática y/o lógica que sirven para describir y verificar sistemas de software y/o hardware.

La especificación **funcional** de un sistema describe *qué* es lo que debe hacer. Los lenguajes de especificación **formal** describen los requerimientos funcionales en términos de fórmulas matemáticas y/ó lógicas.

En la próxima sección introduciremos el lenguaje de especificación a partir del cuál deberá desarrollar el sistema propuesto.

2. Lenguaje de Especificación

Se utilizará un lenguaje de especificación simplificado y acotado a las necesidades del problema a resolver en el experimento.

2.1. Características básicas

- Es un lenguaje tipado, es decir que todas las variables tienen un **tipo** asociado.
- Es un lenguaje basado en lógica de predicados y teoría de conjuntos.
- Se utiliza para especificar máquinas de estado. El estado del sistema **siempre** está *visible* en todas las operaciones.

3. Repaso Teoría de Conjuntos

Un **conjunto** es una *colección* de elementos considerada en si mismo como un objeto.

Denotamos a los conjuntos con llaves:

$$\{elemento1, elemento2, \dots\}$$

El conjunto que no posee elementos se llama **conjunto vacío** y lo notamos: $\emptyset$

Los conjuntos se pueden expresar por **extensión** es decir enumerando todos sus elementos, por ejemplo, el conjunto de los animales domésticos:

$$\{perro, gato, conejo\}$$

Y también pueden expresarse por **comprensión** mediante fórmulas más complejas, por ejemplo para definir el conjunto de números enteros mayores a 100:

$$\{x \in \mathbb{Z} \mid x > 100\}$$

3.1. Pertenencia

Decimos que un elemento **pertenece** a un conjunto cuando de alguna forma está contenido dentro del conjunto.

El operador de pertenencia es $\in$, $x \in X$ indica que x **es un elemento** del conjunto X, para indicar lo contrario, es decir que x **no es un elemento** de X usamos $\notin$.

Por ejemplo, para el conjunto:

$$Alemanes = \{bmw, audi, mercedes, volskwagen\}$$

Trivialmente se puede notar que:

$$audi \in Alemanes$$

Y que:

$$chevrolet \notin Alemanes$$

3.2. Subconjuntos

Un conjunto A **es subconjunto** de un conjunto B si cada elemento de A es también un elemento de B y se denota $A \subseteq B$.

Por ejemplo para el conjunto $Alemanes$ definido anteriormente:

$$\{audi, bmw\} \subseteq Alemanes$$
$$\{audi\} \subseteq Alemanes$$

3.3. Conjunto potencia

El **conjunto potencia** o **partes** de un conjunto, denotado por $\mathbb{P}\, A$, es otro conjunto formado por todos los posibles subconjuntos del conjunto dado.

Por ejemplo, para el conjunto:

$$A = \{1, 2, 3\}$$

El conjunto potencia de A es:

$$\mathbb{P}\, A = \{\{1, 2, 3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1\}, \{2\}, \{3\}, \emptyset\}$$

Notar que $A \in \mathbb{P}\, A$ y $\emptyset \in \mathbb{P}\, A$.

3.4. Operadores

3.4.1. Unión

La unión de dos conjuntos A, B se nota $A \cup B$, determina el conjunto formado por todos los elementos de x tal que $x \in A \vee x \in B$.

Por ejemplo para los conjuntos:

$$A = \{a, b, c, e\}$$
$$B = \{b, d, a, f\}$$

La unión $A \cup B$ resulta:

$$A \cup B = \{a, b, c, e, d, f\}$$

3.4.2. Producto cartesiano

El **producto cartesiano** es una operación entre dos conjuntos A y B que resulta en otro conjunto formado por todos los pares ordenados (a, b) tal que $a \in A$ y $b \in B$. Esta operación se denota como $A \times B$.

Por ejemplo, para los conjuntos:

$$A = \{a, b, c\}$$
$$B = \{1, 2\}$$

Resulta:

$$A \times B = \{(a, 1), (a, 2), (b, 1), (b, 2), (c, 1), (c, 2)\}$$

3.4.3. Proyección

La **proyección** *i-ésima* devuelve el elemento del par ubicado en la posición *i*.

Por ejemplo el par ordenado (a, b) se tiene:

$$(a, b).1 = a$$
$$(a, b).2 = b$$

3.5. Relaciones y Funciones

Una **relación** R es un subconjunto del producto cartesiano entre dos conjuntos.

Matemáticamente, $R \subseteq A \times B$, lo que es igual a decir que $R \in \mathbb{P}(A \times B)$.

Se nota al conjunto de todas las posibles relaciones entre A y B como $A \leftrightarrow B = \mathbb{P}(A \times B)$.

Las **funciones** son relaciones que cumplen con la condición de que a cada elemento del dominio de la relación se le asigna un **único** elemento del rango de la relación.

Es una función:

$$\{(1, a), (2, b)\}$$

No es una función:

$$\{(1, a), (1, b)\}$$

El conjunto de todas las posibles funciones entre A y B se nota $A \to B$ siendo A y B donde A se llama **dominio** de la función y B el **rango**.

3.6. Operadores de funciones

3.6.1. Dominio y Rango

El **dominio** de una función F, es el conjunto conformado por las primeras componentes de los pares ordenados de F y se denota $\operatorname{dom} F$.

De manera análoga, el **rango** de una función F es el conjunto conformado por las segundas componentes de los pares ordenados de F y se denota $\operatorname{ran} F$.

Por ejemplo, para la función $F \in A \to B$ con $A = \{1, 2\}$ y $B = \{4, 5\}$, $F = \{(1, 4), (1, 5)\}$ se tiene:

$$\operatorname{dom} R = \{1\}$$
$$\operatorname{ran} R = \{4, 5\}$$

3.6.2. Restricción de dominio y de rango

Dada una función F y un conjunto A se dice que A restringe el dominio de F ya que la función resultante se compone por los pares de F cuya primer componente se encuentra en el conjunto A y se nota $A \lhd F$, este operador se llama **restricción de dominio**.

Matemáticamente,

$$A \lhd F = \{(x, y) \in F \mid x \in A\}$$

Análogamente, para una función F y un conjunto A, $F \rhd A$ es el operador de **restricción de rango**. T restringe el rango de R ya que el conjunto resultado se compone por los pares de F cuya segunda componente está en A.

Matemáticamente,

$$F \rhd A = \{(x, y) \in F \mid y \in A\}$$

Por ejemplo, para la relación: $R = \{(1, a), (2, b), (3, a), (1, d)\}$ y los conjuntos $S = \{1\}$, $T = \{a\}$:

$$S \lhd R = \{(1, a), (1, d)\}$$
$$R \rhd T = \{(1, a), (3, a)\}$$

3.6.3. Actualizar

Dada una función F y un par ordenado $Q = (a, b)$, el operador actualizar $\oplus$, actualiza la función F agregando el par Q si este no se encuentra en la función o actualizando dicho par si existe algún par ordenado cuya primer componente coincida con la primer componente de Q.

Por ejemplo, para la siguiente función:

$$edades = \{(Federico, 24), (Jorge, 41), (Pablo, 27)\}$$

Se le desea sobrescribir la edad de *Federico* a 25:

$$edades \oplus (Federico, 25) = \{(Federico, 25), (Jorge, 41), (Pablo, 27)\}$$

3.7. Números y aritmética

3.7.1. Enteros

El único conjunto predefinido en el lenguaje de especificación es el conjunto de los enteros $\mathbb{Z}$.

3.7.2. Aritmética

La aritmética del lenguaje de especificación es la usual de los números enteros $+$, $-$, $*$, div, suma, resta, multiplicación y división entera.

3.7.3. Operadores de comparación

Soporta el uso de operadores de comparación para los números enteros $<$, $\leq$, $>$, $\geq$ menor, menor o igual, mayor y mayor o igual.

3.7.4. Cardinalidad

El operador de **cardinalidad** $\#$ enumera la cantidad de elementos que tiene un conjunto dado.

Por ejemplo para el conjunto $A = \{1, 2\}$ se tiene que $\#A = 2$.

4. Tipos

4.1. Declaración de tipos

Existen tres maneras de declarar tipos:

- Sinónimo de tipo
  type $NOMBRE = DEFINICION$
- Tipo enumerado
  enum $NOMBRE = V1 \mid V2 \mid ...$
- Tipo abstracto
  abstract type $NOMBRE$

Los **tipos abstractos** se utilizan para abstraer y simplificar porciones de especificación.

Por ejemplo en la especificación del experimento se utiliza el tipo abstracto $FECHAHORA$ que representa un instante de tiempo dado.

Otra opción sería especificar el tipo $FECHAHORA$ como sinónimo de los enteros o naturales, pero es sabido que la mayoría de los lenguajes de programación poseen una implementación para estos tipos de datos y por ende no justifica agregar complejidad a la especificación.

4.2. Constantes

En el lenguaje de especificación se pueden definir *constantes* globales. Estos valores no pueden modificarse y su valor puede ser consultado desde cualquier operación.

$$\textbf{global}\ NOMBRE : TIPO\,[= VALOR]$$

Las constantes pueden o no tener valor a nivel de especificación pero si deben tenerlo en su implementación.

4.3. Funciones externas

Las funciones externas son funciones cuya especificación puede estar en otro documento de especificación formal o puede tener una implementación trivial. Por ejemplo

$$\textbf{external function}\ DIF\_FECHAS\_DIAS : (FECHAHORA \times FECHAHORA) \to \mathbb{Z}$$

Es una función que retorna la diferencia entre dos fechas en días.

Estas funciones externas pueden ser utilizadas en la especificación de las operaciones del sistema.

5. Estado

El lenguaje de especificación especifica el sistema en término de transiciones de estado definiendo las operaciones que generan dichas transiciones.

El estado del sistema se declara como un conjunto de **variables globales** y la sintaxis utilizada es la siguiente:

$$\begin{aligned}\textbf{StateVariables} ::= & usuarios : DNI \to NOMBRE \\ & claves : DNI \to CLAVE \\ & saldo : MONTO\end{aligned}$$

Donde DNI, $NOMBRE$ y $CLAVE$ son tipos abstractos. El estado se compone de dos funciones que asignan a cada *dni* un *nombre* y una *clave*.

5.1. Estado inicial

La especificación debe definir cuál es el estado inicial del sistema, para eso se utiliza la siguiente sintaxis:

$$\begin{aligned}\textbf{InitialState} ::= & usuarios = \emptyset \\ & claves = \emptyset \\ & saldo = 0\end{aligned}$$

Es decir que ambas funciones comienzan vacías.

6. Operaciones

Las operaciones definen qué es lo que puede hacer el sistema, es decir cómo transiciona el sistema.

Pueden tomar o no argumentos de entrada y declarar o no, variables de salida.

Los argumentos de entrada se define agregando el sufijo **?** a una variable y las variables de salida se le agrega el **!** al final del nombre de la variable.

Una operación se define de la siguiente manera:

$$\begin{aligned}\textbf{op}\ & Carga(dni? : DNI, clave? : CLAVE, saldo? : MONTO, res! : RESULTADO) \mathrel{\widehat{=}} \\ & CargaOK(dni?, clave?, saldo?, res!) \\ & \lor\ UsuarioNoHabilitado(dni?, res!) \\ & \lor\ ClaveIncorrecta(dni?, clave?, res!)\end{aligned}$$

Donde $CargaOK$ es la suboperación que indica el *happy path* de la operación. $UsuarioNoHabilitado$ y $ClaveIncorrecta$ son los caminos que contemplan los posibles errores de la operación.

6.1. Suboperaciones

Las suboperaciones están compuestas por condiciones lógicas que deben ser cumplidas por los parámetros y por el estado actual del sistema.

También deben contener sentencias de asignación que producen la transición de estado.

Por ejemplo la suboperacion $CargaOK$ se especifica de la siguiente manera

$$\begin{aligned}\textbf{sub}\ & CargaOK(dni? : DNI, clave? : CLAVE, saldo? : MONTO, res! : RESULTADO) \mathrel{\widehat{=}} \\ & dni? = administrador \land \\ & claves(dni?) = clave? \land \\ & saldo := saldo + saldo? \land \\ & res! := ok\end{aligned}$$

Las referencias a $administrador$ y ok corresponden a la constante global

$$\textbf{global}\ administrador : DNI$$

y al tipo enumerado

$$\textbf{enum}\ RESULTADO = ok \mid ....$$

Las dos primeras líneas son las condiciones que debe cumplir los parámetros de entrada $dni?$ y $clave?$.

Las últimas dos son asignaciones, $saldo := saldo + saldo?$ asigna un nuevo valor a la variable de estado $saldo$ y $res! := ok$ le asigna el valor ok a la variable de salida $res!$.

6.2. Errores

Por otro lado los errores son muy similares a las suboperaciones con la diferencia de que estos **NO** pueden modificar el estado, es decir que si existe alguna asignación la variable asignada debe ser un parámetro de salida.

Por ejemplo el error $UsuarioNoHabilitado$ se especifica de la siguiente manera

$$\begin{aligned}\textbf{er}\ UsuarioNoHabilitado(dni? : DNI, res! : RESULTADO) \mathrel{\widehat{=}}\\ dni? \neq administrador \land\\ res! := usuarioNoHabilitado\end{aligned}$$

Estas operaciones de error no están asociadas a una sola operación, puede que el error que modela se repita en otras operaciones por lo que puede ser reutilizada en la definición de otras operaciones.