
# Especificación de problemas
## Clase 5

Gustavo Landfried

Grupo Antropocaos

24 de septiembre de 2018

# Objetivo

Detalle instrucciones para que un *mono programador* escriba un programa que calcule el promedio entre siete números.

# Objetivo

Detalle instrucciones para que un *mono programador* escriba un programa que calcule el promedio entre siete números.
Esto nos hace pensar en:

- Roles
- Obligaciones
- Detalles

# Contrato

- La función o el programa que solucione el problema planteado va a ser utilizado por un **usuario**.
- El usuario puede ser otro programador. Normalmente lo será, a veces inclusive puede ser uno mismo.

# Contrato

- La función o el programa que solucione el problema planteado va a ser utilizado por un **usuario**.
- El usuario puede ser otro programador. Normalmente lo será, a veces inclusive puede ser uno mismo.
- Vamos a ver a la especificación de un problema como un **contrato** entre partes:

0. **Programador** de una función que resuelva el problema (usualmente será el rol de ustedes como alumnos).
1. **Usuarios** de esa función (más raro aquí, pero habitual en la vida real).

# Contrato

- La especificación es un **compromiso** entre dos partes

# Contrato

- La especificación es un **compromiso** entre dos partes
- Por ejemplo:
  **Problema**: calcular la raíz cuadrada de un número real

# Contrato

- La especificación es un **compromiso** entre dos partes
- Por ejemplo:
  **Problema**: calcular la raíz cuadrada de un número real
- **Solución**: será una función/programa.
  - Va a recibir un número real como parámetro.
  - Va a calcular un resultado que será un número real.

# Contrato

- La especificación es un **compromiso** entre dos partes
- Por ejemplo:
  **Problema**: calcular la raíz cuadrada de un número real
- **Solución**: será una función/programa.
  - Va a recibir un número real como parámetro.
  - Va a calcular un resultado que será un número real.
- **Restricciones**: para hacer el cálculo, debe recibir un número no negativo.
  - Esto representa la **obligación** de parte del usuario.
  - También representa el **derecho** del implementador/programador de suponer que el argumento **cumple**. En el ejemplo de la raíz cuadrada significa que se recibe un número no negativo.

# Contrato

- La especificación es un **compromiso** entre dos partes
- Por ejemplo:
  **Problema**: calcular la raíz cuadrada de un número real
- **Solución**: será una función/programa.
  - Va a recibir un número real como parámetro.
  - Va a calcular un resultado que será un número real.
- **Restricciones**: para hacer el cálculo, debe recibir un número no negativo.
  - Esto representa la **obligación** de parte del usuario.
  - También representa el **derecho** del implementador/programador de suponer que el argumento **cumple**. En el ejemplo de la raíz cuadrada significa que se recibe un número no negativo.
- **Resultado**: va a ser la raíz cuadrada del número recibido.
  - Esto representa el **compromiso** del programador.
  - Siempre y cuando, se hayan respetado las condiciones. En este caso, que se haya recibido un número real no negativo.
  - El usuario puede suponer que el resultado será correcto.

# Elementos de una especificación

La especificación de un problema tiene las siguientes partes:

- Encabezado:
    0. nombre de la función
    1. lista de parámetros con sus tipos
    2. qué tipo devuelve

# Elementos de una especificación

La especificación de un problema tiene las siguientes partes:

- Encabezado:
    0. nombre de la función
    1. lista de parámetros con sus tipos
    2. qué tipo devuelve
- Precondición:
    0. Condiciones sobre los argumentos de entrada (puede no haber ninguna condición).
    1. Debe ser cumplida por el usuario de la función.
    2. Representa lo que **requiere** la función para hacer su tarea.
    3. Ejemplo: *“el valor de entrada es un real no negativo”*.

# Elementos de una especificación

La especificación de un problema tiene las siguientes partes:

- Encabezado:
    0. nombre de la función
    1. lista de parámetros con sus tipos
    2. qué tipo devuelve
- Precondición:
    0. Condiciones sobre los argumentos de entrada (puede no haber ninguna condición).
    1. Debe ser cumplida por el usuario de la función.
    2. Representa lo que **requiere** la función para hacer su tarea.
    3. Ejemplo: *"el valor de entrada es un real no negativo"*.
- Poscondición:
    0. Condiciones sobre el resultado.
    1. Debe ser cumplida por el programador (i.e. Ustedes).
    2. Representa lo que la función **asegura** que se va a cumplir después de invocarla (si se cumplió la precondición).
    3. Ejemplo: *"la salida es la raíz cuadrada del valor de entrada"*

# Especificación formal

## Objetivo

Cuando pidamos especificar un cierto problema, daremos un enunciado en lenguaje natural y ustedes deberán especificarlo en lenguaje formal. La especificación deberá tener un equilibrio **justo** describiendo *solo* el problema a resolver, no *cómo* se lo resuelve (sobreespecificación). También les daremos especificaciones formales, y ustedes deberán implementar programas que las resuelvan (suponiendo que existe un programa que los hace).

# Especificación formal

## Objetivo

Cuando pidamos especificar un cierto problema, daremos un enunciado en lenguaje natural y ustedes deberán especificarlo en lenguaje formal. La especificación deberá tener un equilibrio **justo** describiendo *solo* el problema a resolver, no *cómo* se lo resuelve (sobreespecificación). También les daremos especificaciones formales, y ustedes deberán implementar programas que las resuelvan (suponiendo que existe un programa que los hace).

- Encabezado
- Precondición
- Poscondición

# Especificación formal

## Objetivo

Cuando pidamos especificar un cierto problema, daremos un enunciado en lenguaje natural y ustedes deberán especificarlo en lenguaje formal. La especificación deberá tener un equilibrio **justo** describiendo *solo* el problema a resolver, no *cómo* se lo resuelve (sobreespecificación). También les daremos especificaciones formales, y ustedes deberán implementar programas que las resuelvan (suponiendo que existe un programa que los hace).

- Encabezado
- Precondición
- Poscondición
- Ejemplo de problema: calcular la raíz cuadrada. Lo llamaremos `rcuad`:

**problema** rcuad(x: $\mathbb{R}$) = result: $\mathbb{R}$ {
**requiere** $x \geq 0$;
**asegura** result * result = x; }

# Especificación formal

## Objetivo

Cuando pidamos especificar un cierto problema, daremos un enunciado en lenguaje natural y ustedes deberán especificarlo en lenguaje formal. La especificación deberá tener un equilibrio **justo** describiendo *solo* el problema a resolver, no *cómo* se lo resuelve (sobreespecificación). También les daremos especificaciones formales, y ustedes deberán implementar programas que las resuelvan (suponiendo que existe un programa que los hace).

- Encabezado
- Precondición
- Poscondición
- Ejemplo de problema: calcular la raíz cuadrada. Lo llamaremos `rcuad`:

**problema** rcuad(x: $\mathbb{R}$) = result: $\mathbb{R}$ {
**requiere** $x \geq 0$;
**asegura** result * result = x; }

# Mecánica

La especificación *funciona* de la siguiente forma:

- El **requiere** me limita el rango de valores que puede tomar mi programa. Si la evaluación del **requiere** da `True`, mi programa debe calcular un resultado.

# Mecánica

La especificación *funciona* de la siguiente forma:

- El **requiere** me limita el rango de valores que puede tomar mi programa. Si la evaluación del **requiere** da `True`, mi programa debe calcular un resultado.

- Si el **requiere** da `True`, ejecuto mi programa y su resultado debe hacer que el **asegura** también dé `True`.

# Mecánica

La especificación *funciona* de la siguiente forma:

- El **requiere** me limita el rango de valores que puede tomar mi programa. Si la evaluación del **requiere** da `True`, mi programa debe calcular un resultado.

- Si el **requiere** da `True`, ejecuto mi programa y su resultado debe hacer que el **asegura** también dé `True`.

- Si el **requiere** o el **asegura** se indefinen, la especificación **no** sirve.

# Mecánica

La especificación *funciona* de la siguiente forma:

- El **requiere** me limita el rango de valores que puede tomar mi programa. Si la evaluación del **requiere** da `True`, mi programa debe calcular un resultado.
- Si el **requiere** da `True`, ejecuto mi programa y su resultado debe hacer que el **asegura** también dé `True`.
- Si el **requiere** o el **asegura** se indefinen, la especificación **no** sirve.
- Si tengo valores de entrada que hacen que el **requiere** diera `True`, pero el **asegura** da `False`... ¡Tengo un **problemón**!

# Mecánica

La especificación *funciona* de la siguiente forma:

- El **requiere** me limita el rango de valores que puede tomar mi programa. Si la evaluación del **requiere** da `True`, mi programa debe calcular un resultado.
- Si el **requiere** da `True`, ejecuto mi programa y su resultado debe hacer que el **asegura** también dé `True`.
- Si el **requiere** o el **asegura** se indefinen, la especificación **no** sirve.
- Si tengo valores de entrada que hacen que el **requiere** diera `True`, pero el **asegura** da `False`... ¡Tengo un **problemón**! Si el valor del parámetro *pasa* el filtro del **requiere**, mi programa debería computar lo indicado por el contrato.

# Definición de predicados

- Es una manera de ayudar en que la especificación que generamos se pueda leer más fácil.
- Darles nombres a predicados (que devuelven `True` o `False`).
- Pueden o no tener variables libres.

# Definición de predicados

- Es una manera de ayudar en que la especificación que generamos se pueda leer más fácil.
- Darles nombres a predicados (que devuelven `True` o `False`).
- Pueden o no tener variables libres.
- Se define con: a) Un predicado b) Un nombre (idealmente representativo); c) Una lista de variables tipadas (que serán las variables libres de la fórmula).

# Definición de predicados

- Es una manera de ayudar en que la especificación que generamos se pueda leer más fácil.
- Darles nombres a predicados (que devuelven `True` o `False`).
- Pueden o no tener variables libres.
- Se define con: a) Un predicado b) Un nombre (idealmente representativo); c) Una lista de variables tipadas (que serán las variables libres de la fórmula).
- Ejemplos:
    0. *"Todo entero positivo es el siguiente de otro"*:

# Definición de predicados

- Es una manera de ayudar en que la especificación que generamos se pueda leer más fácil.
- Darles nombres a predicados (que devuelven `True` o `False`).
- Pueden o no tener variables libres.
- Se define con: a) Un predicado b) Un nombre (idealmente representativo); c) Una lista de variables tipadas (que serán las variables libres de la fórmula).
- Ejemplos:
  0. *"Todo entero positivo es el siguiente de otro"*:
     $A \equiv \{(\forall x : \mathbb{Z})(x > 0 \rightarrow (\exists y : \mathbb{Z})(y + 1 = x))\}$

# Definición de predicados

- Es una manera de ayudar en que la especificación que generamos se pueda leer más fácil.
- Darles nombres a predicados (que devuelven `True` o `False`).
- Pueden o no tener variables libres.
- Se define con: a) Un predicado b) Un nombre (idealmente representativo); c) Una lista de variables tipadas (que serán las variables libres de la fórmula).
- Ejemplos:
  0. *"Todo entero positivo es el siguiente de otro"*:
     $A \equiv \{(\forall x : \mathbb{Z})(x > 0 \rightarrow (\exists y : \mathbb{Z})(y + 1 = x))\}$
  1. *"Dado un número entero positivo, éste es el sucesor de otro"*:

# Definición de predicados

- Es una manera de ayudar en que la especificación que generamos se pueda leer más fácil.
- Darles nombres a predicados (que devuelven `True` o `False`).
- Pueden o no tener variables libres.
- Se define con: a) Un predicado b) Un nombre (idealmente representativo); c) Una lista de variables tipadas (que serán las variables libres de la fórmula).
- Ejemplos:
  0. *"Todo entero positivo es el siguiente de otro"*:
     $A \equiv \{(\forall x : \mathbb{Z})(x > 0 \rightarrow (\exists y : \mathbb{Z})(y + 1 = x))\}$
  1. *"Dado un número entero positivo, éste es el sucesor de otro"*:
     $B(x : \mathbb{Z}) \equiv \{(x > 0 \rightarrow (\exists y : \mathbb{Z})(y + 1 = x))\}$

**Atención**

La lista de variables del predicado (abusando un poco, nos referiremos como parámetros) tiene que coincidir con las apariciones libres en la fórmula.

# Definición de funciones

Los predicados son casos particulares de funciones, son de tipo $\mathbb{B}$.

- Los definimos con:
  0. Un nombre (nombres representativos!).
  1. Un término, generalmente con variables libres.
  2. Una lista de variables tipadas (aparecen libres en el cuerpo).

# Definición de funciones

Los predicados son casos particulares de funciones, son de tipo $\mathbb{B}$.

- Los definimos con:
  0. Un nombre (nombres representativos!).
  1. Un término, generalmente con variables libres.
  2. Una lista de variables tipadas (aparecen libres en el cuerpo).

0. *"Una función que defina el sucesor de un número entero"*:
   $\text{Suc}(x : \mathbb{Z}) \equiv \{x + 1\}$
   En los ejemplos anteriores, quedaría:
   - $A' \equiv \{(\forall x : \mathbb{Z})(x > 0 \rightarrow (\exists y : \mathbb{Z})(\text{Suc}(y) = x))\}$

# Definición de funciones

Los predicados son casos particulares de funciones, son de tipo $\mathbb{B}$.

- Los definimos con:
  0. Un nombre (nombres representativos!).
  1. Un término, generalmente con variables libres.
  2. Una lista de variables tipadas (aparecen libres en el cuerpo).

0. *"Una función que defina el sucesor de un número entero"*:
   $\text{Suc}(x : \mathbb{Z}) \equiv \{x + 1\}$
   En los ejemplos anteriores, quedaría:
   - $A' \equiv \{(\forall x : \mathbb{Z})(x > 0 \to (\exists y : \mathbb{Z})(\text{Suc}(y) = x))\}$
   - $B'(x : \mathbb{Z}) \equiv \{(x > 0 \to (\exists y : \mathbb{Z})(\text{Suc}(y) = x))\}$
1. *"El polinomio $2x^3 + 5x^2 - x + 7$ tiene (por lo menos) dos raíces reales"*:
   - Definimos la función $\text{Polinomio}(x : \mathbb{R}) \equiv \{2.x^3 + 5.x^2 - x + 7\}$
   - Y la usamos en $\text{DosRaices} \equiv \{(\exists i, j : \mathbb{R})(i \neq j \wedge \text{Polinomio}(i) = 0 \wedge \text{Polinomio}(j) = 0)\}$

# Sumatorias y productorias

- Sirven para formar otros términos
- Se definen sobre intervalos finitos:
  - SumaDePrimeros($k : \mathbb{Z}$) $\equiv \{\Sigma_{i=1}^{k} i\}$

# Sumatorias y productorias

- Sirven para formar otros términos
- Se definen sobre intervalos finitos:
  - SumaDePrimeros($k : \mathbb{Z}$) $\equiv \{\Sigma_{i=1}^{k} i\}$
  - Factorial($k : \mathbb{Z}$) $\equiv \{\Pi_{i=1}^{k} i\}$

# Sumatorias y productorias

- Sirven para formar otros términos
- Se definen sobre intervalos finitos:
  - SumaDePrimeros($k : \mathbb{Z}$) $\equiv \{\Sigma_{i=1}^{k} i\}$
  - Factorial($k : \mathbb{Z}$) $\equiv \{\Pi_{i=1}^{k} i\}$
- Sumatoria de rango vacío se define como 0.
- Productoria de rango vacío se define como 1.
- Ambas se indefinen si algún término es indefinido.
- Se utilizan mucho combinadas con $\beta$ (pasaba de `False`/`True` a 0/1) para contar elementos que cumplen con una propiedad (por ejemplo, cantidad de números pares de una lista).

# Pensemos un poco

Veamos qué cosas tendríamos que considerar para especificar el siguiente problema:

> `jugar(a)`: toma un mazo de cartas `a` y va sacando cartas del mismo, asignándoselas a un jugador. Debe dejar de tomar cartas si la suma de los valores obtenidos es mayor o igual a 21. Devuelve la lista de cartas obtenidas por el jugador. Además, el mazo de cartas no se debe modificar al llamar a esta función.

# Pensemos un poco

Veamos qué cosas tendríamos que considerar para especificar el siguiente problema:

> `jugar(a)`: toma un mazo de cartas `a` y va sacando cartas del mismo, asignándoselas a un jugador. Debe dejar de tomar cartas si la suma de los valores obtenidos es mayor o igual a 21. Devuelve la lista de cartas obtenidas por el jugador. Además, el mazo de cartas no se debe modificar al llamar a esta función.

- **requiere**: ¿Qué condiciones/propiedades tienen los datos de entrada?
- **asegura**: ¿Qué vamos a hacer? Es decir, ¿Qué propiedades va a cumplir la salida?