Introducción a la Computación (Matemática)

Primer Cuatrimestre de 2018

Especificación de Problemas

Especificación, algoritmo y programa

Especificación de un problema:

- ¿Qué problema tenemos?
- Lenguaje formal (ej. lógica de primer orden).

Especificación, algoritmo y programa

Especificación de un problema:

- ¿Qué problema tenemos?
- Lenguaje formal (ej. lógica de primer orden).

Algoritmo: Una solución abstracta del problema (escrita para humanos).

- ¿Cómo resolvemos el problema?
- Pseudocódigo.

Especificación, algoritmo y programa

Especificación de un problema:

- ¿Qué problema tenemos?
- Lenguaje formal (ej. lógica de primer orden).

Algoritmo: Una solución abstracta del problema (escrita para humanos).

- ¿Cómo resolvemos el problema?
- Pseudocódigo.

Programa: Una solución concreta del problema (escrita para computadoras).

- ¿Cómo resuelve la computadora el problema?
- Lenguaje de programación (ej. C^{++}, Python).

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).

Algoritmo

Un **algoritmo** es una secuencia **finita** de instrucciones que, al ejecutarlas en el orden preestablecido, permite hallar la solución a un problema dado.

Algoritmo

Un **algoritmo** es una secuencia **finita** de instrucciones que, al ejecutarlas en el orden preestablecido, permite hallar la solución a un problema dado.

El algoritmo es un concepto matemático bastante anterior a las computadoras. La palabra "algoritmo" deriva del nombre del matemático persa al-Juarismi (780-850).

Algoritmo

Un **algoritmo** es una secuencia **finita** de instrucciones que, al ejecutarlas en el orden preestablecido, permite hallar la solución a un problema dado.

El algoritmo es un concepto matemático bastante anterior a las computadoras. La palabra “algoritmo” deriva del nombre del matemático persa al-Juarismi (780-850).

No todos los programas son correctas implementaciones de algoritmos. Los algoritmos deben terminar siempre pero algunos programas pueden no parar nunca.

Programa

Un programa es una **secuencia finita de SENTENCIAS**: asignaciones, condicionales, ciclos, etc.

Ejemplo:

```
int fil = 5;
while (fil >= 1) {
   int col = 1;
   while (col <= fil) {
      cout << col << " ";
      col = col + 1;
   }
   cout << endl;
   fil = fil - 1;
}
```

Programa vs. Algoritmo

Ejemplo:

1. Moje el cabello.
2. Coloque shampoo.
3. Masajee suavemente y deje actuar por 2 min.
4. Enjuague.
5. Repita el procedimiento (desde 1).

Programa vs. Algoritmo

Ejemplo:

1. Moje el cabello.
2. Coloque shampoo.
3. Masajee suavemente y deje actuar por 2 min.
4. Enjuague.
5. Repita el procedimiento (desde 1) una vez.

¿Qué ocurre cuando hay errores en los programas?

El 31/12/2008 todos los dispositivos MS ZUNE se colgaron al mismo tiempo. ¿Por qué?


```
year = ORIGINYEAR; /* = 1980 */
while (days > 365) {
    if (IsLeapYear(year)) {
        if (days > 366) {
            days = days - 366;
            year = year + 1;
        }
    } else {
        days = days - 365;
        year = year + 1;
    }
}
```

Especificación de problemas. 1

Primero recordemos que nuestro modelo de cómputo es una máquina de cambio de estado. Donde esos estados son representados por los valores que toman las variables que describen un estado en un momento dado.

Especificación de problemas. 1

Primero recordemos que nuestro modelo de cómputo es una máquina de cambio de estado. Donde esos estados son representados por los valores que toman las variables que describen un estado en un momento dado.

Programar, en un sentido abstracto, es dar la secuencia de sentencias que permiten transformar el estado inicial en uno final. Así, la ejecución de un programa puede verse como la secuencia de estado que este genera.

Especificación de problemas. 1

Primero recordemos que nuestro modelo de cómputo es una máquina de cambio de estado. Donde esos estados son representados por los valores que toman las variables que describen un estado en un momento dado.

Programar, en un sentido abstracto, es dar la secuencia de sentencias que permiten transformar el estado inicial en uno final. Así, la ejecución de un programa puede verse como la secuencia de estado que este genera.

Especificar, es establecer condiciones para pasar de un estado a otro. Esencialmente es predicar sobre las variables de estado.

Especificación de problemas. 2

La especificación de un problema se describe en un lenguaje formal:

- Precondición: indicando cuáles son los valores de las variables de entrada aceptables, y
- Poscondición: qué propiedades deben cumplir los valores de las variables de la salida.

Especificación de problemas. 2

La especificación de un problema se describe en un lenguaje formal:

- Precondición: indicando cuáles son los valores de las variables de entrada aceptables, y
- Poscondición: qué propiedades deben cumplir los valores de las variables de la salida.

Eso se interpreta de la siguiente forma: para todo estado que satisfaga la precondición, el algoritmo debe terminar exitosamente en un estado que satisfaga las propiedades especificadas en la poscondición.

Especificación de problemas. 3

Pero, ¿para qué especificar?

- Antes de sentarnos a programar, debemos tener bien claro **qué** tenemos que resolver.
  - Error común: programar sin un objetivo claro.

Especificación de problemas. 3

Pero, ¿para qué especificar?

- Antes de sentarnos a programar, debemos tener bien claro **qué** tenemos que resolver.
  - Error común: programar sin un objetivo claro.

- Especificar es el primer paso para después poder probar la **correctitud** de un algoritmo.
  - Un algoritmo **correcto respecto de la especificación** ejecuta una transformación un estado inicial en otro final donde ambos cumplan las propiedades de la especificación.

Definición (Especificación) de un problema

Encabezado: nombre: $A_1 \in T_1 \times \cdots \times A_n \in T_n \to T_{RV}$
Precondición: $\{\ P\ \}$
Poscondición: $\{\ Q\ \}$

- *nombre*: nombre que le damos al problema
  - será resuelto por una función con ese mismo nombre
- Lista de argumentos:
  - Cada A_i es el nombre del argumeto.
  - Cada T_i es el tipo de datos del argumento.
  - T_{RV} es el tipo del dato de salida.
- P y Q son predicados, denominados la precondición y la poscondición del problema/función.

Argumentos

Los argumentos se comportan como variables.
Como tales, pueden ser modificadas libremente en el algoritmo.

Argumentos

Los argumentos se comportan como variables.
Como tales, pueden ser modificadas libremente en el algoritmo.

En la precondición le ponemos nombre a sus valores iniciales:

Precondición: $\{x = x_0 \wedge y = y_0\}$

Argumentos

Los argumentos se comportan como variables.
Como tales, pueden ser modificadas libremente en el algoritmo.

En la precondición le ponemos nombre a sus valores iniciales:

Precondición: $\{x = x_0 \wedge y = y_0\}$

En la poscondición predicamos sobre esos valores iniciales:

Poscondición: $\{RV = x_0 + y_0\}$

Argumentos

Los argumentos se comportan como variables.
Como tales, pueden ser modificadas libremente en el algoritmo.

En la precondición le ponemos nombre a sus valores iniciales:

Precondición: $\{x = x_0 \wedge y = y_0\}$

En la poscondición predicamos sobre esos valores iniciales:

Poscondición: $\{RV = x_0 + y_0\}$

Si queremos que una variable de entrada cumpla una propiedad al finalizar el algoritmo, lo especificamos en la poscondición:

Poscondición: $\{RV = x0 + y0 \ \wedge \ x = 8\}$

Ejemplo

Problema “sumaCuadrados”: dados dos números enteros, devolver la suma de los cuadrados de ambos.

Ejemplo

Problema "sumaCuadrados": dados dos números enteros, devolver la suma de los cuadrados de ambos.

Encabezado: $sumaCuadrados : a \in \mathbb{Z} \times b \in \mathbb{Z} \rightarrow \mathbb{Z}$
Precondición: $\{a = a_0 \wedge b = b_0\}$
Poscondición: $\{RV = a_0 * a_0 + b_0 * b_0\}$

RV es el valor de retorno del problema.

Correctitud.

- **Contrato:** *El programador escribe P tal que si se suministran datos que hacen verdadera la precondición, entonces P termina en una cantidad finita de pasos con un valor que hace verdadera la postcondición.*

Correctitud.

- **Contrato:** *El programador escribe P tal que si se suministran datos que hacen verdadera la precondición, entonces P termina en una cantidad finita de pasos con 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.

Correctitud.

- **Contrato:** *El programador escribe P tal que si se suministran datos que hacen verdadera la precondición, entonces P termina en una cantidad finita de pasos con 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?

Correctitud.

- **Contrato:** *El programador escribe P tal que si se suministran datos que hacen verdadera la precondición, entonces P termina en una cantidad finita de pasos con 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?

Lenguaje de especificación

- Tipos de datos: $\mathbb{Z}$ (enteros), $\cdot[\,]$ (arreglos), $\mathbb{B} = \{true, false\}$ (booleanos, o valores de verdad).
  - Ej: $n \in \mathbb{Z}$, $x \in \mathbb{R}$, $L \in \mathbb{Z}[\,]$, $b \in \mathbb{B}$.
- Un **término** puede ser una variable, una constante o una operación aplicada a otros términos.
  - Ej: 0, $0 + 1$, $n - 1$, true, $a \wedge b$.
- Un **predicado** es un enunciado sobre términos, que puede tomar valor verdadero o falso.
  - Ej: $x > 0$, $i + 1 = 0 \wedge j - 2 \geq 0$, $2 > 3$.

Lenguaje de especificación (cont.)

- Operaciones booleanas (conectivos lógicos): $\neg \wedge \vee \Rightarrow$
  - Ej: $p \wedge q$, $(\neg a \vee b) \Rightarrow c$.
- Operaciones de $\mathbb{Z}$: $+ - * / |\cdot|$; $< = \leq > \geq$...
  - Ej: $a + b$, $|-5| < 5$.
- Operaciones de arreglos: $\cdot[\cdot]$, $|\cdot|$ (longitud).
  - Ej: $A[3]$, $|A|$.
- Funciones matemáticas auxiliares.
  - Ej: $f(x \in \mathbb{R}) = \begin{cases} 0 & \text{si} \quad x \leq 0 \\ x + 1 & \text{si} \quad x > 0. \end{cases}$
- Predicados auxiliares.
  - Ej: $Par(n \in \mathbb{Z}) \equiv (\exists k \in \mathbb{Z})\, n = 2 * k$.

Lenguaje de especificación (cont.)

- Cuantificadores: $\forall$ (universal), $\exists$ (existencial).
  - $(\forall i \in \mathbb{Z})\ 0 \leq i < |A| \Rightarrow Par(A[i])$
    Todos los elementos del arreglo A son pares.
  - $(\exists i \in \mathbb{Z})\ 0 \leq i < |A| \wedge Par(A[i])$
    Existe algún elemento par en el arreglo A.

Lenguaje de especificación (cont.)

- Cuantificadores: $\forall$ (universal), $\exists$ (existencial).
  - $(\forall i \in \mathbb{Z})\ 0 \leq i < |A| \Rightarrow Par(A[i])$
    Todos los elementos del arreglo A son pares.
  - $(\exists i \in \mathbb{Z})\ 0 \leq i < |A| \wedge Par(A[i])$
    Existe algún elemento par en el arreglo A.
- Acumuladores: $\sum$ (sumatoria), $\#$ (contador).
  - $RV = \sum_{i \in \mathbb{Z}\ /\ 0 \leq i < |A| \wedge Par(A[i])} A[i]$
    La suma de los elementos pares del arreglo A.
  - $RV = (\#i \in \mathbb{Z})\ 0 \leq i < |A| \wedge Par(A[i])$
    La cantidad de elementos pares en el arreglo A.

Especificación de problemas

Las especificaciones son abstractas.

Por lo tanto, no modelan los problemas de representación (p.ej., *overflow*).

- `int` $\rightsquigarrow \mathbb{Z}$

Dichos problemas de representación deberán ser tenidos en cuenta durante la implementación del algoritmo, en un lenguaje de programación concreto.

Ejemplo

Problema “div”: dados dos números enteros, devolver la división entera del primero por el segundo.

Ejemplo

Problema "div": dados dos números enteros, devolver la división entera del primero por el segundo.

Encabezado: $div : a \in \mathbb{Z} \times b \in \mathbb{Z} \to \mathbb{Z}$

Precondición: $\{a = a_0 \wedge b = b_0 \wedge b_0 \neq 0\}$

Poscondición: $\{(\exists r \in \mathbb{Z})\ 0 \leq r < |b_0| \wedge a_0 = RV * b_0 + r\}$

Problemas sin valor de retorno

Un problema puede no tener un valor de retorno.

Ejemplo:
Problema "inc": dado un entero, sumarle uno.

Problemas sin valor de retorno

Un problema puede no tener un valor de retorno.

Ejemplo:
Problema "inc": dado un entero, sumarle uno.

Encabezado: $inc : x \in \mathbb{Z} \rightarrow \emptyset$
Precondición: $\{x = x_0\}$
Poscondición:

Problemas sin valor de retorno

Un problema puede no tener un valor de retorno.

Ejemplo:
Problema "inc": dado un entero, sumarle uno.

Encabezado: $inc : x \in \mathbb{Z} \rightarrow \emptyset$
Precondición: $\{x = x_0\}$
Poscondición: $\{x = x_0 + 1\}$

Notar que no se menciona RV en la poscondición.

Problemas sin valor de retorno

Otro ejemplo:

Problema "incTodos": dado un arreglo de enteros, suma uno a todos sus elementos.

Problemas sin valor de retorno

Otro ejemplo:

Problema "incTodos": dado un arreglo de enteros, suma uno a todos sus elementos.

Encabezado: $incTodos : A \in \mathbb{Z}[] \rightarrow \emptyset$

Precondición: $\{A = A_0\}$

Poscondición:

Problemas sin valor de retorno

Otro ejemplo:

Problema "incTodos": dado un arreglo de enteros, suma uno a todos sus elementos.

Encabezado: $incTodos : A \in \mathbb{Z}[] \rightarrow \emptyset$

Precondición: $\{A = A_0\}$

Poscondición: $\{|A| = |A_0| \wedge$
$(\forall i \in \mathbb{Z})\ 0 \leq i < |A| \Rightarrow A[i] = A_0[i] + 1\}$

Problemas sin valor de retorno

Otro ejemplo:

Problema "incTodos": dado un arreglo de enteros, suma uno a todos sus elementos.

Encabezado: $incTodos : A \in \mathbb{Z}[] \rightarrow \emptyset$

Precondición: $\{A = A_0\}$

Poscondición: $\{|A| = |A_0| \wedge (\forall i \in \mathbb{Z})\ 0 \leq i < |A| \Rightarrow A[i] = A_0[i] + 1\}$

Problema "swap": dados dos números enteros, intercambiarlos.

Encabezado: $swap : a \in \mathbb{Z} \times b \in \mathbb{Z} \rightarrow \emptyset$

Precondición: $\{a = a_0 \wedge b = b_0\}$

Poscondición: $\{a = b_0 \wedge b = a_0\}$

Problemas sin valor de retorno

Otro ejemplo:

Problema "swapPrimUlt": dada una lista de enteros con al menos un elemento, intercambiar el primero y el último.

Problemas sin valor de retorno

Otro ejemplo:

Problema "swapPrimUlt": dada una lista de enteros con al menos un elemento, intercambiar el primero y el último.

Encabezado: $swapPrimUlt : L \in \mathbb{Z}[] \rightarrow \emptyset$

Precondición: $\{|L| > 0 \wedge L = L_0\}$

Poscondición: $\{|L| = |L_0| \wedge$
$L[0] = L_0[|L| - 1] \ \wedge \ L[|L| - 1] = L_0[0] \wedge$
$(\forall i \in \mathbb{Z})\ 1 \leq i < |L| - 1 \Rightarrow L[i] = L_0[i]\}$

Más ejemplos

Problema "distinto": dado un número entero, dar otro distinto.

Encabezado: $distinto : a \in \mathbb{Z} \to \mathbb{Z}$
Precondición: $\{a = a_0\}$
Poscondición: $\{RV \neq a_0 \wedge RV > 0\}$

Más ejemplos

Problema "distinto": dado un número entero, dar otro distinto.

Encabezado: $distinto : a \in \mathbb{Z} \to \mathbb{Z}$
Precondición: $\{a = a_0\}$
Poscondición: $\{RV \neq a_0 \wedge RV > 0\}$

Notar que habrá algoritmos que en términos reales resuelven el problema, pero que no satisfacen la postcondición.

Más ejemplos

Problema "distinto": dado un número entero, dar otro distinto.

Encabezado: $distinto : a \in \mathbb{Z} \to \mathbb{Z}$
Precondición: $\{a = a_0\}$ sobreespecificación
Poscondición: $\{RV \neq a_0 \wedge RV > 0\}$

Notar que habrá algoritmos que en términos reales resuelven el problema, pero que no satisfacen la postcondición.

Encabezado: $distinto : a \in \mathbb{Z} \to \mathbb{Z}$
Precondición: $\{a = a_0 \wedge a_0 > 0\}$ subespecificación
Poscondición: $\{RV \neq a_0\}$

Más ejemplos

Problema "distinto": dado un número entero, dar otro distinto.

Encabezado: $distinto : a \in \mathbb{Z} \to \mathbb{Z}$
Precondición: $\{a = a_0\}$ sobreespecificación
Poscondición: $\{RV \neq a_0 \wedge RV > 0\}$

Notar que habrá algoritmos que en términos reales resuelven el problema, pero que no satisfacen la postcondición.

Encabezado: $distinto : a \in \mathbb{Z} \to \mathbb{Z}$
Precondición: $\{a = a_0 \wedge a_0 > 0\}$ subespecificación
Poscondición: $\{RV \neq a_0\}$

Notar que en este caso habrá algoritmos que satisfacen la especificación pero no son solución del problema real.

Sobre-especificación

Sobre-especificación

- Consiste en dar una postcondición más restrictiva que lo que se necesita, o bien dar una precondición más laxa.

Sobre-especificación

- Consiste en dar una postcondición más restrictiva que lo que se necesita, o bien dar una precondición más laxa.
- Limita los posibles algoritmos que resuelven el problema, porque impone más condiciones para la salida, o amplía los datos de entrada.

Sobre-especificación

- Consiste en dar una postcondición más restrictiva que lo que se necesita, o bien dar una precondición más laxa.
- Limita los posibles algoritmos que resuelven el problema, porque impone más condiciones para la salida, o amplía los datos de entrada.
- Ejemplo:

  Encabezado: $distinto : a \in \mathbb{Z} \to \mathbb{Z}$
  Precondición: $\{a = a_0\}$
  Poscondición: $\{RV \neq a_0 \wedge RV > 0\}$

- ... en lugar de:

  Encabezado: $distinto : a \in \mathbb{Z} \to \mathbb{Z}$
  Precondición: $\{a = a_0\}$
  Poscondición: $\{RV \neq a_0\}$

Sub-especificación

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

Sub-especificación

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

Encabezado: $distinto : a \in \mathbb{Z} \to \mathbb{Z}$
Precondición: $\{a = a_0 \land a_0 > 0\}$
Poscondición: $\{RV \neq a_0\}$

... en vez de:

Encabezado: $distinto : a \in \mathbb{Z} \to \mathbb{Z}$
Precondición: $\{a = a_0\}$
Poscondición: $\{RV \neq a_0\}$

Propiedades de una especificación Pre/Pos

Asumamos que $\{P\}\ A\ \{Q\}$, significa que el algoritmo A es correcto respecto de la precondición P y la poscondición Q. Entonces si se cumple que $\{P\}\ A\ \{Q\}$, tendremos que:

1. Si $P' \Rightarrow P$ entonces resulta que $\{P'\}\ A\ \{Q\}$.
2. Si $Q \Rightarrow Q'$ entonces resulta que $\{P\}\ A\{Q'\}$.

Propiedades de una especificación Pre/Pos

Asumamos que $\{P\}\ A\ \{Q\}$, significa que el algoritmo A es correcto respecto de la precondición P y la poscondición Q. Entonces si se cumple que $\{P\}\ A\ \{Q\}$, tendremos que:

1. Si $P' \Rightarrow P$ entonces resulta que $\{P'\}\ A\ \{Q\}$.
2. Si $Q \Rightarrow Q'$ entonces resulta que $\{P\}\ A\{Q'\}$.

Más aún, si se cumple que: $\{P_1\}\ A\ \{Q_1\}$ y $\{P_2\}\ A\ \{Q_2\}$. Entonces:

1. $\{P_1 \wedge P_2\}\ A\ \{Q_1 \wedge Q_2\}$.
2. $\{P_1 \vee P_2\}\ A\ \{Q_1 \vee Q_2\}$.

Ejemplo

Encontrar la raíz cuadrada entera de un número entero.
Ejemplos: $raizEntera(8) \to 2$, $raizEntera(9) \to 3$.

Encabezado: $raizEntera : n \in \mathbb{Z} \to \mathbb{Z}$
Precondición: $\{n = n_0\}$
Poscondición: $\{RV * RV \leq n_0 < (RV + 1) * (RV + 1)\}$

¿Está bien?

Ejemplo

Encontrar la raíz cuadrada entera de un número entero.
Ejemplos: $raizEntera(8) \rightarrow 2$, $raizEntera(9) \rightarrow 3$.

Encabezado: $raizEntera : n \in \mathbb{Z} \rightarrow \mathbb{Z}$
Precondición: $\{n = n_0\}$
Poscondición: $\{RV * RV \leq n_0 < (RV + 1) * (RV + 1)\}$

¿Está bien? No, porque no hay solución cuando $n_0 < 0$. La especificación **no puede ser satisfecha** por un algoritmo, porque la precondición es demasiado débil (ejemplo de sobreespecificación).

Ejemplo

Encontrar la raíz cuadrada entera de un número entero.
Ejemplos: $raizEntera(8) \to 2$, $raizEntera(9) \to 3$.

Encabezado: $raizEntera : n \in \mathbb{Z} \to \mathbb{Z}$
Precondición: $\{n = n_0\}$
Poscondición: $\{RV * RV \leq n_0 < (RV + 1) * (RV + 1)\}$

¿Está bien? No, porque no hay solución cuando $n_0 < 0$. La especificación **no puede ser satisfecha** por un algoritmo, porque la precondición es demasiado débil (ejemplo de sobreespecificación).

Agregamos entonces $n_0 \geq 0$ a la precondición.

Ejemplo

Encontrar la raíz cuadrada entera de un número entero.
Ejemplos: $raizEntera(8) \to 2$, $raizEntera(9) \to 3$.

Encabezado: $raizEntera : n \in \mathbb{Z} \to \mathbb{Z}$
Precondición: $\{n = n_0\}$
Poscondición: $\{RV * RV \leq n_0 < (RV + 1) * (RV + 1)\}$

¿Está bien? No, porque no hay solución cuando $n_0 < 0$. La especificación **no puede ser satisfecha** por un algoritmo, porque la precondición es demasiado débil (ejemplo de sobreespecificación).

Agregamos entonces $n_0 \geq 0$ a la precondición.

El lenguaje de especificación es más expresivo que el modelo de cómputo: nos permite especificar **problemas sin solución**.

Especificar problemas sin solución

- La conjetura de Goldbach dice que todo entero par mayor que 2 puede ser expresado como la suma de dos números primos.
- **Especificar:**

Especificar problemas sin solución

- La conjetura de Goldbach dice que todo entero par mayor que 2 puede ser expresado como la suma de dos números primos.
- **Especificar:**
- Encabezado: $goldbach : \emptyset \rightarrow \mathbb{B}$
  Precondición: $\{true\}$
  Poscandición: $\{RV = (\forall n : \mathbb{Z})(n > 2 \wedge n \text{ mod } 2 = 0 \rightarrow (\exists p : \mathbb{Z})(\exists q : \mathbb{Z})(esPrimo(p) \wedge esPrimo(q) \wedge n = p+q))\}$

Especificar problemas sin solución

- La conjetura de Goldbach dice que todo entero par mayor que 2 puede ser expresado como la suma de dos números primos.
- **Especificar:**
- Encabezado: $goldbach : \emptyset \to \mathbb{B}$
  Precondición: $\{true\}$
  Poscandición: $\{RV = (\forall n : \mathbb{Z})(n > 2 \wedge n \text{ mod } 2 = 0 \ \to (\exists p : \mathbb{Z})(\exists q : \mathbb{Z})(esPrimo(p) \wedge esPrimo(q) \wedge n = p + q))\}$
- Observar que estamos especificando el problema. Es posible que en este punto no sepamos cómo vamos a resolver este problema (si es que se puede resolver!), y es bueno no pensar en eso al momento de especificar.

Otro ejemplo de especificación

Encabezado: $sumarPares : A \in \mathbb{Z}[] \rightarrow \mathbb{Z}$

Precondición: $\{A = A_0\}$

Poscondición: $\{RV = \sum_{0 \leq i < |A_0| \wedge Par(A_0[i])} A_0[i]\}$

donde $Par(n) \equiv (\exists k)\, n = 2 * k$

Obs: Si no se especifica el tipo de una variable, por defecto es $\mathbb{Z}$.

Otro ejemplo de especificación

Encabezado: $sumarPares : A \in \mathbb{Z}[] \to \mathbb{Z}$
Precondición: $\{A = A_0\}$
Poscondición: $\{RV = \sum_{0 \leq i < |A_0| \wedge Par(A_0[i])} A_0[i]\}$

donde $Par(n) \equiv (\exists k)\, n = 2 * k$

Obs: Si no se especifica el tipo de una variable, por defecto es $\mathbb{Z}$.

Lo siguiente es pensar un algoritmo que satisfaga esta especificación. Lo escribimos usando pseudocógido.

Pseudocódigo

C++	**Pseudocódigo**
a = b;	a $\leftarrow$ b
if (cond) {..} else {..}	if (cond) {..} else {..}
while (cond) {..}	while (cond) {..}
! && ||	$\neg$ $\wedge$ $\vee$
== >= <=	$=$ $\geq$ $\leq$
% /	mod div
return exp;	$RV \leftarrow exp$ (*)

Pseudocódigo

C++	**Pseudocódigo**
a = b;	a $\leftarrow$ b
if (cond) {..} else {..}	if (cond) {..} else {..}
while (cond) {..}	while (cond) {..}
! && ||	$\neg$ $\wedge$ $\vee$
== >= <=	$=$ $\geq$ $\leq$
% /	mod div
return exp;	$RV \leftarrow exp$ (*)

(*) En C++, "return exp;" termina la ejecución de la función. En nuestro pseudocógido, "$RV \leftarrow exp$" es una simple asignación, tras la cual el algoritmo continúa.

Ejemplo de algoritmo en pseudocógido

Encabezado: $sumarPares : A \in \mathbb{Z}[] \rightarrow \mathbb{Z}$

Precondición: $\{A = A_0\}$

Poscondición: $\{RV = \sum_{0 \leq i < |A_0| \wedge Par(A_0[i])} A_0[i]\}$

Ejemplo de algoritmo en pseudocógido

Encabezado: $sumarPares : A \in \mathbb{Z}[] \rightarrow \mathbb{Z}$
Precondición: $\{A = A_0\}$
Poscondición: $\{RV = \sum_{0 \leq i < |A_0| \wedge Par(A_0[i])} A_0[i]\}$

$RV \leftarrow 0$
$i \leftarrow 0$
while $(i < |A|)$ {
    if $(A[i]$ mod $2 = 0)$ {
        $RV \leftarrow RV + A[i]$
    }
    $i \leftarrow i + 1$
}

donde $i \in \mathbb{Z}$.

Ejemplo de algoritmo en pseudocógido

Encabezado: $sumarPares : A \in \mathbb{Z}[] \to \mathbb{Z}$

Precondición: $\{A = A_0\}$

Poscondición: $\{RV = \sum_{0 \leq i < |A_0| \wedge Par(A_0[i])} A_0[i]\}$

$RV \leftarrow 0$
$i \leftarrow 0$
while $(i < |A|)$ {
    if $(A[i] \bmod 2 = 0)$ {
        $RV \leftarrow RV + A[i]$
    }
    $i \leftarrow i + 1$
}

donde $i \in \mathbb{Z}$.

¿Cómo sabemos si este algoritmo es correcto respecto de la especificación?

Testing vs. correctitud de algoritmos

Sean E una especificación de un problema y S un algoritmo.

Testing de S: Experimentación de la corrida de S para un conjunto finito de datos de entrada, donde verificamos si cumple E.

Testing vs. correctitud de algoritmos

Sean E una especificación de un problema y S un algoritmo.

Testing de S: Experimentación de la corrida de S para un conjunto finito de datos de entrada, donde verificamos si cumple E.

Correctitud de S respecto de E: Demostración formal de que el algoritmo S transforma todos los posibles datos de entrada, en salidas de acuerdo con E.

Correctitud de algoritmos

Terna de Hoare

$$\{\text{precondición}\}$$
$$\text{algoritmo}$$
$$\{\text{poscondición}\}$$

Correctitud de algoritmos

Terna de Hoare

$\{$precondición$\}$	$\{P\}$
algoritmo	S
$\{$poscondición$\}$	$\{Q\}$

Correctitud de algoritmos

Terna de Hoare

$\{$precondición$\}$	$\{P\}$
algoritmo	S
$\{$poscondición$\}$	$\{Q\}$

S modifica al estado P, ¿pero lleva a Q?

Correctitud de algoritmos

Terna de Hoare

$$\begin{array}{cc} \{\text{precondición}\} & \{P\} \\ \text{algoritmo} & S \\ \{\text{poscondición}\} & \{Q\} \end{array}$$

S modifica al estado P, ¿pero lleva a Q?

Una forma de probarlo: Ver que la poscondición más fuerte de ejecutar S a partir de P, implica Q.

$$¿sp(S, P) \Rightarrow Q?$$

$sp(S, P)$ (*strongest postcondition*) es el predicado más fuerte que resulta de ejecutar S a partir del estado P.

Asignación

$\{P\}\ x\ \leftarrow E\ \{Q\}$

Queremos probar que $sp(x \leftarrow E,\ P) \Rightarrow Q$.

Asignación

$\{P\}\ x \leftarrow E\ \{Q\}$

Queremos probar que $sp(x \leftarrow E,\ P) \Rightarrow Q$.

Definición:

$$sp(x \leftarrow E,\ P) \equiv (\exists v)\ x = E[x : v] \wedge P[x : v]$$

donde:

- v es una variable no usada;
- $H[x : E]$ es la sustitución de cada instancia en H de la variable x por la expresión E.
  Ejemplo: $(x + y)[y : z^2 + 1] = (x + z^2 + 1)$.

Asignación: Ejemplo

Probar la correctitud del siguiente algoritmo respecto de su especificación:

$$P \equiv \{x > 1\}$$
$$x \leftarrow x + 1$$
$$Q \equiv \{x > 2\}$$

Asignación: Ejemplo

Probar la correctitud del siguiente algoritmo respecto de su especificación:

$$P \equiv \{x > 1\}$$
$$x \leftarrow x + 1$$
$$Q \equiv \{x > 2\}$$

$$\begin{aligned} &sp(x \leftarrow x + 1, x > 1) \equiv \\ &\quad \equiv (\exists v)\ x = (x + 1)[x : v] \wedge (x > 1)[x : v] \\ &\quad \equiv (\exists v)\ x = v + 1 \wedge v > 1 \\ &\quad \Rightarrow x > 2 \end{aligned}$$

Secuencialización

$\{P\}\ S_1;S_2\ \{Q\}$

Queremos probar que $sp(S_1;S_2,\ P) \Rightarrow Q$.

Secuencialización

$\{P\}\ S_1;S_2\ \{Q\}$

Queremos probar que $sp(S_1;S_2,\ P) \Rightarrow Q$.

Definición:

$$sp(S_1;S_2,\ P) \equiv sp(S_2, sp(S_1, P))$$

Secuencialización: Ejemplo

$$P \equiv \{x > 1\}$$
$$x \leftarrow x + 1$$
$$y \leftarrow 2 * x$$
$$Q \equiv \{y > 4\}$$

Secuencialización: Ejemplo

$$P \equiv \{x > 1\}$$
$$x \leftarrow x + 1$$
$$y \leftarrow 2 * x$$
$$Q \equiv \{y > 4\}$$

$$\begin{aligned}
&sp(x \leftarrow x + 1; y \leftarrow 2 * x, P) \equiv \\
&\quad \equiv sp(y \leftarrow 2 * x, sp(x \leftarrow x + 1, P)) \\
&\quad \equiv sp(y \leftarrow 2 * x, (\exists a)\ x = (x + 1)[x : a] \wedge (x > 1)[x : a]) \\
&\quad \equiv sp(y \leftarrow 2 * x, (\exists a)\ x = a + 1 \wedge a > 1) \\
&\quad \equiv (\exists b)\ y = (2 * x)[y : b] \wedge ((\exists a)\ x = a + 1 \wedge a > 1)[y : b] \\
&\quad \equiv (\exists b)\ y = 2 * x \wedge (\exists a)\ x = a + 1 \wedge a > 1 \\
&\quad \equiv y = 2 * x \wedge (\exists a)\ x = a + 1 \wedge a > 1 \\
&\quad \Rightarrow y = 2 * x \wedge x > 2 \\
&\quad \Rightarrow y > 4
\end{aligned}$$

Condicional

$\{P\}$ if (B) S_1 else S_2 $\{Q\}$

Queremos probar que $sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \Rightarrow Q$.

Condicional

$\{P\}$ if (B) S_1 else S_2 $\{Q\}$

Queremos probar que $sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \Rightarrow Q$.

Definición:

$$sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \equiv sp(S_1, B \wedge P) \vee sp(S_2, \neg B \wedge P)$$

Condicional

$\{P\}$ if (B) S_1 else S_2 $\{Q\}$

Queremos probar que $sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \Rightarrow Q$.

Definición:

$$sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \equiv sp(S_1, B \wedge P) \vee sp(S_2, \neg B \wedge P)$$

¿Qué pasa si no hay "else"?

Condicional

$\{P\}$ if (B) S_1 else S_2 $\{Q\}$

Queremos probar que $sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \Rightarrow Q$.

Definición:

$$sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \equiv sp(S_1, B \wedge P) \vee sp(S_2, \neg B \wedge P)$$

¿Qué pasa si no hay "else"? Formalmente, if (B) S_1 equivale a if (B) S_1 else pass, donde pass es una instrucción especial que no tiene efecto:

$$sp(\text{pass},\ P) \equiv P$$

Condicional

$\{P\}$ if (B) S_1 else S_2 $\{Q\}$

Queremos probar que $sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \Rightarrow Q$.

Definición:

$$sp(\text{if } (B) \ S_1 \text{ else } S_2,\ P) \equiv sp(S_1, B \wedge P) \vee sp(S_2, \neg B \wedge P)$$

¿Qué pasa si no hay "else"? Formalmente, if (B) S_1 equivale a if (B) S_1 else pass, donde pass es una instrucción especial que no tiene efecto:

$$sp(\text{pass},\ P) \equiv P$$

Por lo tanto:

$$sp(\text{if } (B) \ S_1,\ P) \equiv sp(S_1, B \wedge P) \vee (\neg B \wedge P)$$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$
if $(a \leq b)$ {
  $RV \leftarrow 0$
} else {
  $RV \leftarrow a - b$
}
$Q \equiv \{RV = a_0 \dot{-} b_0\}$

donde $x \dot{-} y = \begin{cases} 0 & \text{si} \quad x \leq y \\ x - y & \text{si} \quad x > y \end{cases}$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$
if $(a \leq b)$ $\{$ $RV \leftarrow 0$ $\}$ else $\{$ $RV \leftarrow a - b$ $\}$
$Q \equiv \{RV = a_0 \dot{-} b_0\}$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$
if $(a \leq b)$ { $RV \leftarrow 0$ } else { $RV \leftarrow a - b$ }
$Q \equiv \{RV = a_0 \dot{-} b_0\}$

$sp(if(a \leq b)\{RV \leftarrow 0\}$ else $\{RV \leftarrow a - b\}, P) \equiv$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$
if $(a \leq b)$ { $RV \leftarrow 0$ } else { $RV \leftarrow a - b$ }
$Q \equiv \{RV = a_0 \dot{-} b_0\}$

$sp(if(a \leq b)\{RV \leftarrow 0\} \text{ else } \{RV \leftarrow a - b\}, P) \equiv$
$\equiv sp(RV \leftarrow 0, \quad a \leq b \wedge P) \vee$
$sp(RV \leftarrow a - b, \; a > b \wedge P)$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$
if $(a \leq b)$ { $RV \leftarrow 0$ } else { $RV \leftarrow a - b$ }
$Q \equiv \{RV = a_0 \dot{-} b_0\}$

$$
\begin{aligned}
& sp(if(a \leq b)\{RV \leftarrow 0\} \text{ else } \{RV \leftarrow a - b\}, P) \equiv \\
& \quad \equiv sp(RV \leftarrow 0, \quad a \leq b \wedge P) \vee \\
& \qquad sp(RV \leftarrow a - b, \; a > b \wedge P) \\
& \quad \equiv ((\exists x)\, RV = 0[RV : x] \qquad \wedge (a \leq b \wedge P)[RV : x]) \vee \\
& \qquad ((\exists y)\, RV = (a - b)[RV : y] \; \wedge (a > b \wedge P)[RV : y])
\end{aligned}
$$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$

if $(a \leq b)$ { $RV \leftarrow 0$ } else { $RV \leftarrow a - b$ }

$Q \equiv \{RV = a_0 \dot{-} b_0\}$

$$
\begin{aligned}
& sp(if(a \leq b)\{RV \leftarrow 0\} \text{ else } \{RV \leftarrow a - b\}, P) \equiv \\
& \quad \equiv sp(RV \leftarrow 0, \quad a \leq b \wedge P) \vee \\
& \qquad sp(RV \leftarrow a - b, \; a > b \wedge P) \\
& \quad \equiv ((\exists x)\, RV = 0[RV : x] \quad \wedge (a \leq b \wedge P)[RV : x]) \vee \\
& \qquad ((\exists y)\, RV = (a - b)[RV : y] \; \wedge (a > b \wedge P)[RV : y]) \\
& \quad \equiv ((\exists x)\, RV = 0 \quad \wedge (a \leq b \wedge P)) \vee \\
& \qquad ((\exists y)\, RV = (a - b) \; \wedge (a > b \wedge P))
\end{aligned}
$$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$
if $(a \leq b)$ { $RV \leftarrow 0$ } else { $RV \leftarrow a - b$ }
$Q \equiv \{RV = a_0 \dot{-} b_0\}$

$$
\begin{aligned}
& sp(if(a \leq b)\{RV \leftarrow 0\} \text{ else } \{RV \leftarrow a - b\}, P) \equiv \\
& \quad \equiv sp(RV \leftarrow 0, \quad a \leq b \wedge P) \vee \\
& \qquad sp(RV \leftarrow a - b, \; a > b \wedge P) \\
& \quad \equiv ((\exists x)\, RV = 0[RV : x] \quad \wedge (a \leq b \wedge P)[RV : x]) \vee \\
& \qquad ((\exists y)\, RV = (a - b)[RV : y] \; \wedge (a > b \wedge P)[RV : y]) \\
& \quad \equiv ((\exists x)\, RV = 0 \quad \wedge (a \leq b \wedge P)) \vee \\
& \qquad ((\exists y)\, RV = (a - b) \; \wedge (a > b \wedge P)) \\
& \quad \equiv (RV = 0 \quad \wedge (a \leq b \wedge a = a_0 \wedge b = b_0)) \vee \\
& \qquad (RV = (a - b) \; \wedge (a > b \wedge a = a_0 \wedge b = b_0))
\end{aligned}
$$

Condicional: Ejemplo

$P \equiv \{a = a_0 \wedge b = b_0\}$
if $(a \leq b)$ { $RV \leftarrow 0$ } else { $RV \leftarrow a - b$ }
$Q \equiv \{RV = a_0 \dot{-} b_0\}$

$$
\begin{aligned}
& sp(if(a \leq b)\{RV \leftarrow 0\} \text{ else } \{RV \leftarrow a - b\}, P) \equiv \\
& \quad \equiv sp(RV \leftarrow 0, \quad a \leq b \wedge P) \vee \\
& \qquad sp(RV \leftarrow a - b, \; a > b \wedge P) \\
& \quad \equiv ((\exists x)\, RV = 0[RV : x] \qquad \wedge (a \leq b \wedge P)[RV : x]) \vee \\
& \qquad ((\exists y)\, RV = (a - b)[RV : y] \; \wedge (a > b \wedge P)[RV : y]) \\
& \quad \equiv ((\exists x)\, RV = 0 \qquad \wedge (a \leq b \wedge P)) \vee \\
& \qquad ((\exists y)\, RV = (a - b) \; \wedge (a > b \wedge P)) \\
& \quad \equiv (RV = 0 \qquad \wedge (a \leq b \wedge a = a_0 \wedge b = b_0)) \vee \\
& \qquad (RV = (a - b) \; \wedge (a > b \wedge a = a_0 \wedge b = b_0)) \\
& \quad \Rightarrow RV = a_0 \dot{-} b_0
\end{aligned}
$$

Repaso de la clase de hoy

- Especificación de problemas.
- Algoritmo vs. programa.
- Encabezado, precondición y poscondición.
- Lenguaje de especificación.
- Sobre- y subespecificación.
- Noción de correctitud de algoritmos.
- Pseudocódigo.
- Testing vs. correctitud de algoritmos.
- Terna de Hoare; poscondición más fuerte (sp).
- Asignación, secuencialización, condicional.
- Correctitud de asignación, secuencialización y condicional.

Próximos temas

- Mucha ejercitación.
- Correctitud de ciclos.