

Algoritmos y Estructuras de Datos I

Primer cuatrimestre de 2018

Departamento de Computación - FCEyN - UBA

Introducción a la especificación de problemas
Lógica proposicional

1

Algoritmos y Estructuras de Datos I

Objetivo: Aprender a programar en **lenguajes imperativos**.

- ▶ **Especificar** problemas.
 - ▶ Describirlos en lenguaje formal.
- ▶ Pensar **algoritmos** para resolver los problemas.
 - ▶ En esta materia nos concentramos en programas para **tratamiento de secuencias**.
- ▶ **Razonar** acerca de estos algoritmos.
 - ▶ Obtener una visión abstracta del cómputo.
 - ▶ Definir un manejo simbólico y herramientas para demostrar propiedades de nuestros programas.
 - ▶ Probar **corrección** de un programa respecto de su especificación.

3

Algoritmos y Estructuras de Datos I

Régimen de aprobación

- ▶ Parciales
 - ▶ Dos parciales (LU con Álgebra I firmada)
 - ▶ Dos recuperatorios (al final de la cursada)
- ▶ Trabajos prácticos
 - ▶ Dos entregas + coloquios
 - ▶ Dos recuperatorios
 - ▶ Grupos de cuatro alumnos
- ▶ Examen final o un coloquio (en caso de tener dado el final de Álgebra I al finalizar la cursada)

4

campus.exactas.uba.ar

algo1-tm-alu@dc.uba.ar
algo1-tm-doc@dc.uba.ar

algo1-tt-alu@dc.uba.ar
algo1-tt-doc@dc.uba.ar

5

Bibliografía

- ▶ ¿Cómo pensamos y programamos?
 - ▶ **The Science of Programming**. David Gries. Springer Verlag, 1981
 - ▶ **Reasoned programming**. K. Broda, S. Eisenbach, H. Khoshnevisan, S. Vickers. Prentice-Hall, 1994
- ▶ Testing de Programas
 - ▶ **Software Testing and Analysis: Process, Principles and Techniques**. M. Pezze, M. Young, Wiley, 2007.
- ▶ Referencia del lenguaje que usaremos
 - ▶ **The C++ Programming Language, 4th Edition**. B. Stroustrup. Addison-Wesley Professional, 2003

6

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 escrito en un lenguaje de programación.

7

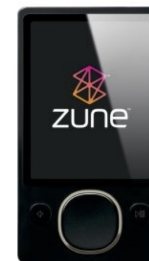
Especificación de problemas

- ▶ 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).

8

¿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;  
    }  
}
```

9

Definición de un problema

```
proc nombre(parámetros){  
  Pre { P }  
  Post { Q }  
}
```

- **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, entrada y salida)
 - Nombre del parámetro
 - Tipo de datos del parámetro
- P y Q son predicados, denominados la **precondición** y la **postcondición** del procedimiento.

10

Ejemplos

```
proc rcuad(in x : ℝ, out result : ℝ) {  
  Pre {  $x \geq 0$  }  
  Post {  $result * result = x$  }  
}
```

```
proc suma(in x : ℤ, in y : ℤ, out result : ℤ) {  
  Pre { True }  
  Post {  $result = x + y$  }  
}
```

```
proc resta(in x : ℤ, in y : ℤ, out result : ℤ) {  
  Pre { True }  
  Post {  $result = x - y$  }  
}
```

```
proc cualquieramayor(in x : ℤ, out result : ℤ) {  
  Pre { True }  
  Post {  $result > x$  }  
}
```

11

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?

12

Otro ejemplo

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

```
proc cociente(in dividendo : ℤ, in divisor : ℤ, out result : ℤ) {  
  Pre {  $divisor > 0$  }  
  Post {  
     $result * divisor \leq dividendo$   
     $\wedge (result + 1) * divisor > dividendo$   
  }  
}
```

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?

13

Argumentos que se modifican (inout)

Problema: Incrementar en 1 el argumento de entrada.

- Alternativa sin modificar la entrada (usual).

```
proc incremento(in a :  $\mathbb{Z}$ , out result :  $\mathbb{Z}$ ){  
  Pre { True }  
  Post { result = a + 1 }  
}
```

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

```
problema incremento-modificando(inout a :  $\mathbb{Z}$ ){  
  Pre { a = a0 }  
  Post { a = a0 + 1 }  
}
```

- La variable a_0 es una **metavariable**, que representa el valor inicial de la variable a , y que usamos en la postcondición para relacionar el valor de salida de a con su valor inicial.

14

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

15

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:

```
proc distinto(in x :  $\mathbb{Z}$ , out result :  $\mathbb{Z}$ ) {  
  Pre { True }  
  Post { result = x + 1 }  
}
```

- ... en lugar de:

```
proc distinto(in x :  $\mathbb{Z}$ , out result :  $\mathbb{Z}$ ){  
  Pre { True }  
  Post { result  $\neq$  x }  
}
```

16

Sub-especificación

- Consiste en dar 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 :  $\mathbb{Z}$ , out result :  $\mathbb{Z}$ ){  
  Pre { x > 0 }  
  Post { result  $\neq$  x }  
}
```

... en vez de:

```
proc distinto(in x :  $\mathbb{Z}$ , out result :  $\mathbb{Z}$ ){  
  Pre { True }  
  Post { result  $\neq$  x }  
}
```

17

Especificar los siguientes problemas

- ▶ Encontrar una raíz de un polinomio de grado 2.

```
proc raiz(in a : ℝ, in b : ℝ, in c : ℝ, out result : ℝ){  
  Pre { True }  
  Post { a * result * result + b * result + c = 0 }  
}
```
- ▶ Está bien especificado? **No!** Existen datos de entrada que cumplen la precondition para los cuales no se puede escribir un programa que termine cumpliendo la postcondición.
- ▶

```
proc raiz(in a : ℝ, in b : ℝ, in c : ℝ, out result : ℝ){  
  Pre { a ≠ 0 ∧ b * b ≥ 4 * a * c }  
  Post { a * result * result + b * result + c = 0 }  
}
```

18

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

19

Tipos de datos de nuestro lenguaje de especificación

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

20

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 con mayúsculas.
- ▶ $\text{ord}(a)$ da la posición del elemento en la definición (empezando de 0).
- ▶ Inversa: El nombre del tipo funciona como inversa de ord .

21

Ejemplo de tipo enumerado

Definimos el tipo Día así:

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

► Valen:

- $\text{ord}(\text{LUN}) = 0$
- $\text{Día}(2) = \text{MIE}$
- $\text{JUE} < \text{VIE}$

22

Tipo upla (o tupla)

- Uplas, de dos o más elementos, cada uno de cualquier tipo.
- $T_0 \times T_1 \times \dots \times T_k$: Tipo de las k -uplas de elementos de tipos T_0, T_1, \dots, 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.
- *nésimo*: $(a_0, \dots, a_k)_m$ es el valor a_m en caso de que $0 \leq m \leq k$. Si no, está indefinido.
- Ejemplos:
 - $(7, 5)_0 = 7$
 - $('a', \text{Domingo}, 78)_2 = 78$

23

Funciones y predicados auxiliares

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

```
fun  $f$ (argumentos) : tipo = expresion;
```

```
pred  $p$ (argumentos){formula}
```

- 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).
- Ejemplo:

```
fun suc( $x : \mathbb{Z}$ ) :  $\mathbb{Z} = x + 1$ ;
```

24

Ejemplos de funciones auxiliares

- `fun e() : $\mathbb{R} = 2,7182$;`
 - `fun inverso($x : \mathbb{R}$) : $\mathbb{R} = 1/x$;`
 - `pred par($n : \mathbb{Z}$) { (n mod 2) = 0 }`
 - `pred impar($n : \mathbb{Z}$) { \neg (par(n)) }`
 - `pred esFinde($d : \text{Día}$) { $d = \text{SAB} \vee d = \text{DOM}$ }`
- Otra forma:
- ```
pred esFinde2($d : \text{Día}$) { $d > \text{VIE}$ }
```

25

## Expresiones condicionales

Función que elige entre dos elementos del mismo tipo, según una condición booleana (guarda)

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

Por ejemplo

- ▶ expresión que devuelve el máximo entre dos elementos:

```
fun máx(a, b : ℤ) : ℤ = IfThenElseFi(ℤ)(a > b, a, b);
```

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

```
fun máx(a, b : ℤ) : ℤ = IfThenElseFi(a > b, a, b);
```

o bien

```
fun máx(a, b : ℤ) : ℤ = if a > b then a else b fi;
```

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

```
fun unoSobre(x : ℝ) : ℝ = if x ≠ 0 then 1/x else 0 fi;
```

no se indefine cuando  $x = 0$

26

## 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 precondition y la postcondition con predicados de primer orden.
- ▶ No podemos usar otros problemas en la especificación (dado que no pertenecen a la lógica de primer orden). Sí podemos usar predicados y funciones auxiliares ya definidos.

27

## Lógica proposicional - Sintaxis

- ▶ Símbolos:

true, false,  $\neg$ ,  $\wedge$ ,  $\vee$ ,  $\rightarrow$ ,  $\leftrightarrow$ , (, )

- ▶ Variables proposicionales (infinitas)

$p, q, r, \dots$

- ▶ Fórmulas

1. true y false son fórmulas
2. Cualquier variable proposicional es una fórmula
3. Si  $A$  es una fórmula,  $\neg A$  es una fórmula
4. Si  $A_1, A_2, \dots, A_n$  son fórmulas,  $(A_1 \wedge A_2 \wedge \dots \wedge A_n)$  es una fórmula
5. Si  $A_1, A_2, \dots, A_n$  son fórmulas,  $(A_1 \vee A_2 \vee \dots \vee A_n)$  es una fórmula
6. Si  $A$  y  $B$  son fórmulas,  $(A \rightarrow B)$  es una fórmula
7. Si  $A$  y  $B$  son fórmulas,  $(A \leftrightarrow B)$  es una fórmula

28

## Semántica clásica

- ▶ Dos valores de verdad: "verdadero" (T) y "falso" (F).
- ▶ Conociendo el valor de las variables proposicionales de una fórmula, podemos calcular el valor de verdad de la fórmula.

| $p$ | $\neg p$ |
|-----|----------|
| T   | F        |
| F   | T        |

| $p$ | $q$ | $(p \wedge q)$ |
|-----|-----|----------------|
| T   | T   | T              |
| T   | F   | F              |
| F   | T   | F              |
| F   | F   | F              |

| $p$ | $q$ | $(p \vee q)$ |
|-----|-----|--------------|
| T   | T   | T            |
| T   | F   | T            |
| F   | T   | T            |
| F   | F   | F            |

| $p$ | $q$ | $(p \rightarrow q)$ |
|-----|-----|---------------------|
| T   | T   | T                   |
| T   | F   | F                   |
| F   | T   | T                   |
| F   | F   | T                   |

| $p$ | $q$ | $(p \leftrightarrow q)$ |
|-----|-----|-------------------------|
| T   | T   | T                       |
| T   | F   | F                       |
| F   | T   | F                       |
| F   | F   | T                       |

29

## Tautologías, contradicciones y contingencias

- Una fórmula es una **tautología** si siempre toma el valor  $T$  para valores definidos de sus variables proposicionales.

Por ejemplo,  $((p \wedge q) \rightarrow p)$  es tautología:

| $p$ | $q$ | $(p \wedge q)$ | $((p \wedge q) \rightarrow p)$ |
|-----|-----|----------------|--------------------------------|
| T   | T   | T              | T                              |
| T   | F   | F              | T                              |
| F   | T   | F              | T                              |
| F   | F   | F              | T                              |

- Una fórmula es una **contradicción** si siempre toma el valor  $F$  para valores definidos de sus variables proposicionales.

Por ejemplo,  $(p \wedge \neg p)$  es contradicción:

| $p$ | $\neg p$ | $(p \wedge \neg p)$ |
|-----|----------|---------------------|
| T   | F        | F                   |
| F   | T        | F                   |

- Una fórmula es una **contingencia** cuando no es ni tautología ni contradicción.

30

## Equivalencias entre fórmulas

- **Teorema.** Las siguientes son tautologías.

1. Idempotencia

$$(p \wedge p) \leftrightarrow p$$

$$(p \vee p) \leftrightarrow p$$

2. Asociatividad

$$(p \wedge q) \wedge r \leftrightarrow p \wedge (q \wedge r)$$

$$(p \vee q) \vee r \leftrightarrow p \vee (q \vee r)$$

3. Conmutatividad

$$(p \wedge q) \leftrightarrow (q \wedge p)$$

$$(p \vee q) \leftrightarrow (q \vee p)$$

4. Distributividad

$$p \wedge (q \vee r) \leftrightarrow (p \wedge q) \vee (p \wedge r)$$

$$p \vee (q \wedge r) \leftrightarrow (p \vee q) \wedge (p \vee r)$$

5. Reglas de De Morgan

$$\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$$

$$\neg(p \vee q) \leftrightarrow \neg p \wedge \neg q$$

31

## Relación de fuerza

- Decimos que  **$A$  es más fuerte que  $B$**  cuando  $(A \rightarrow B)$  es tautología.

- También decimos que  **$A$  fuerza a  $B$**  o que  **$B$  es más débil que  $A$** .

- Por ejemplo,

1. ¿ $(p \wedge q)$  es más fuerte que  $p$ ?    sí

2. ¿ $(p \vee q)$  es más fuerte que  $p$ ?    no

3. ¿ $p$  es más fuerte que  $(q \rightarrow p)$ ?    sí

Pero notemos que si  $q$  está indefinido y  $p$  es verdadero entonces  $(q \rightarrow p)$  está indefinido.

4. ¿ $p$  es más fuerte que  $q$ ?    no

5. ¿ $p$  es más fuerte que  $p$ ?    sí

6. ¿hay una fórmula más fuerte que todas?    false!

7. ¿hay una fórmula más débil que todas?    true

32

## Expresión bien definida

- Toda expresión está **bien definida** en un estado si todas las proposiciones valen  $T$  o  $F$ .

- Sin embargo, existe la posibilidad de que haya expresiones que no estén bien definidas.

- Por ejemplo, la expresión  $x/y$  no está bien definida si  $y = 0$ .

- Por esta razón, necesitamos una lógica que nos permita decir que está bien definida la siguiente expresión

- $y = 0 \vee x/y = 5$

- Para esto, introducimos **tres** valores de verdad:

1. verdadero ( $T$ )

2. falso ( $F$ )

3. indefinido ( $\perp$ )

33



## Semántica trivaluada (secuencial)

Se llama **secuencial** porque ...

- ▶ los términos se evalúan de izquierda a derecha,
- ▶ la evaluación termina cuando se puede deducir el valor de verdad, aunque el resto esté indefinido.

Introducimos los operadores lógicos  $\wedge_L$  (y-luego, o *conditional and*, o **cand**),  $\vee_L$  (o-luego o *conditional or*, o **cor**).

| $p$     | $q$     | $(p \wedge_L q)$ |
|---------|---------|------------------|
| T       | T       | T                |
| T       | F       | F                |
| F       | T       | F                |
| F       | F       | F                |
| T       | $\perp$ | $\perp$          |
| F       | $\perp$ | F                |
| $\perp$ | T       | $\perp$          |
| $\perp$ | F       | $\perp$          |
| $\perp$ | $\perp$ | $\perp$          |

| $p$     | $q$     | $(p \vee_L q)$ |
|---------|---------|----------------|
| T       | T       | T              |
| T       | F       | T              |
| F       | T       | T              |
| F       | F       | F              |
| T       | $\perp$ | T              |
| F       | $\perp$ | $\perp$        |
| $\perp$ | T       | $\perp$        |
| $\perp$ | F       | $\perp$        |
| $\perp$ | $\perp$ | $\perp$        |

34

## Semántica trivaluada (secuencial)

¿Cuál es la tabla de verdad de  $\rightarrow_L$ ?

| $p$     | $q$     | $(p \rightarrow_L q)$ |
|---------|---------|-----------------------|
| T       | T       | T                     |
| T       | F       | F                     |
| F       | T       | T                     |
| F       | F       | T                     |
| T       | $\perp$ | $\perp$               |
| F       | $\perp$ | T                     |
| $\perp$ | T       | $\perp$               |
| $\perp$ | F       | $\perp$               |
| $\perp$ | $\perp$ | $\perp$               |

35

## Cuantificadores

El lenguaje de especificación provee formas de predicar sobre los elementos de un tipo de datos

- ▶  $(\forall x : T) P(x)$ : Término de tipo Bool. Afirma que todos los elementos de tipo  $T$  cumplen la propiedad  $P$ .
  - ▶ Se lee “Para todo  $x$  de tipo  $T$  se cumple  $P(x)$ ”
- ▶  $(\exists x : T) P(x)$ : Término de tipo Bool. Afirma que al menos un elemento de tipo  $T$  cumple la propiedad  $P$ .
  - ▶ Se lee “Existe al menos un  $x$  de tipo  $T$  que cumple  $P(x)$ ”

En la expresión  $(\forall x : T) P(x)$ , la variable  $x$  está **ligada** al cuantificador. Una variable es **libre** cuando no está ligada a ningún cuantificador.

36

## Ejemplo

- ▶ **Ejemplo:** Crear un predicado `esPrimo` que sea **Verdadero** si y sólo si el número  $n$  es un número primo.
- ▶ `pred esPrimo( $n : \mathbb{Z}$ ) {`  
 $n > 1 \wedge (\forall n' : \mathbb{Z})(1 < n' < n \rightarrow n \bmod n' \neq 0)$   
`}`
- ▶ **Ejemplo:** Especificar el problema de, dado un número mayor a 1, indicar si el número es un número primo.
- ▶ `proc primo(in  $n : \mathbb{Z}$ , out  $result : \text{Bool}$ ) {`  
`Pre { $n > 1$ }`  
`Post { $result = \text{esPrimo}(n)$ }`  
`}`

37

## Operando con cuantificadores

- **Ejemplo:** Todos los enteros entre 1 y 10 son pares:  
 $(\forall n : \mathbb{Z})(1 \leq n \leq 10 \rightarrow n \bmod 2 = 0).$
- **Ejemplo:** Existe un entero entre 1 y 10 que es par:  
 $(\exists n : \mathbb{Z})(1 \leq n \leq 10 \wedge n \bmod 2 = 0).$
- En general, si queremos decir que todos los enteros  $x$  que cumplen  $P(x)$  también cumplen  $Q(x)$ , decimos:  
 $(\forall x : \mathbb{Z})(P(x) \rightarrow Q(x)).$
- Para decir que existe un entero que cumple  $P(x)$  y que también cumple  $Q(x)$ , decimos:  
 $(\exists x : \mathbb{Z})(P(x) \wedge Q(x)).$

38

## Operando con cuantificadores

- La **negación** de un cuantificador universal es un cuantificador existencial, y viceversa:  
 $\neg(\forall n : \mathbb{Z})P(n) \leftrightarrow (\exists n : \mathbb{Z})\neg P(n).$   
 $\neg(\exists n : \mathbb{Z})P(n) \leftrightarrow (\forall n : \mathbb{Z})\neg P(n).$
- Un cuantificador universal **generaliza la conjunción**:  
 $(\forall n : \mathbb{Z})(a \leq n \leq b \rightarrow P(n)) \wedge P(b+1)$   
 $\leftrightarrow (\forall n : \mathbb{Z})(a \leq n \leq b+1 \rightarrow P(n)).$
- Un cuantificador existencial generaliza la disyunción:  
 $(\exists n : \mathbb{Z})(a \leq n \leq b \wedge P(n)) \vee P(b+1)$   
 $\leftrightarrow (\exists n : \mathbb{Z})(a \leq n \leq b+1 \wedge P(n)).$

39

## Especificar problemas

- La **conjetura de Goldbach** dice que todo entero par mayor que 2 puede ser expresado como la suma de dos números primos.
- **Ejemplo:** Especificar un procedimiento que indique si esta conjetura es verdadera.
- ```
proc goldbach(out result : Bool) {  
  Pre { True }  
  Post { result = true  $\leftrightarrow$   $(\forall n : \mathbb{Z})(n > 2 \wedge n \bmod 2 = 0 \rightarrow$   
     $(\exists p : \mathbb{Z})(\exists q : \mathbb{Z})(\text{esPrimo}(p) \wedge \text{esPrimo}(q) \wedge n = p + q))$  }  
}
```
- 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.

40

Especificar problemas

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

41

## Bibliografía

- ▶ David Gries - The Science of Programming
  - ▶ Chapter 1 - Propositions (Fórmulas, Tautologías, etc.)
  - ▶ Chapter 2 - Reasoning using Equivalence Transformations (Propiedades, De Morgan, etc.)