

Algoritmos y Estructuras de Datos I

Primer cuatrimestre de 2023

Departamento de Computación - FCEyN - UBA

Lógica proposicional

1

Habíamos visto...

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

- ▶ **Especificar** problemas.
 - ▶ Describirlos en un lenguaje semiformal.
- ▶ Pensar **algoritmos** para resolver los problemas.
 - ▶ En esta materia nos concentramos en programas para **tratamiento de secuencias** principalmente.
- ▶ Empezar a **razonar** acerca de estos algoritmos y programas.
 - ▶ Veremos conceptos de testing.

2

Definición (Especificación) de un problema

```
problema nombre(parámetros) :tipo de dato del resultado{  
  requiere etiqueta { condiciones sobre los parámetros de entrada }  
  asegura etiqueta { condiciones sobre los parámetros de salida }  
}
```

- ▶ **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:
 - ▶ Nombre del parámetro
 - ▶ Tipo de datos del parámetro
- ▶ **tipo de dato del resultado**: tipo de dato del resultado del problema (inicialmente especificaremos funciones)
 - ▶ En los asegura, podremos referenciar el valor devuelto con el nombre de **res**
- ▶ **etiquetas**: son nombres **opcionales** que nos servirán para nombrar declarativamente a las condiciones de los requiere o asegura.

3

Definición (Especificación) de un problema

- ▶ **Sobre los requiere**
 - ▶ Describen todas las condiciones y posibles valores o casuísticas de los parámetros de entrada.
 - ▶ Puede haber más de un requiere (recomendamos una condición por renglón). Se asume que valen todos juntos (es una conjunción).
 - ▶ Evitar contradicciones (un requiere no debería contradecir a otro).
- ▶ **Sobre los asegura**
 - ▶ Describen todas las condiciones y posibles valores o casuísticas de los parámetros de salida y entrada/salida en función de los parámetros de entrada.
 - ▶ Puede haber más de un asegura (recomendamos una condición por renglón). Se asume que valen todos juntos (es una conjunción).
 - ▶ Evitar contradicciones (un asegura no debería contradecir a otro).

4

Antes de continuar... hablemos de lógica proposicional

- ▶ Si bien no utilizaremos un lenguaje formal para especificar... ¿Es lo mismo decir...?
 - ▶ Mañana llueve e iré a comprar un paraguas
 - ▶ Si mañana llueve iré a comprar un paraguas
 - ▶ O mañana no llueve o no iré a comprar un paraguas
 - ▶ Compraré un paraguas por si mañana llueve
 - ▶ Si compro un paraguas, mañana llueve

5

Lógica proposicional

- ▶ Es la lógica que habla sobre las proposiciones.
- ▶ Son oraciones que tienen un valor de verdad, Verdadero o Falso (aunque vamos a usar una variación).
- ▶ Sirve para poder deducir el valor de verdad de una proposición, a partir de conocer el valor de otras.

6

Lógica proposicional - Sintaxis

- ▶ Símbolos:
 $\text{True}, \text{False}, \neg, \wedge, \vee, \rightarrow, \leftrightarrow, (,)$
- ▶ 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

7

Ejemplos

¿Cuáles son fórmulas?

- ▶ $p \vee q$ no
- ▶ $(p \vee q)$ sí
- ▶ $p \vee q \rightarrow r$ no
- ▶ $(p \vee q) \rightarrow r$ no
- ▶ $((p \vee q) \rightarrow r)$ sí
- ▶ $(p \rightarrow q \rightarrow r)$ no

8

Semántica clásica

- Dos valores de verdad: “verdadero” (V) y “falso” (F).
- Interpretación:
 - True siempre vale V.
 - False siempre vale F.
 - \neg se interpreta como “no”, se llama **negación**.
 - \wedge se interpreta como “y”, se llama **conjunción**.
 - \vee se interpreta como “o” (no exclusivo), se llama **disyunción**.
 - \rightarrow se interpreta como “si... entonces”, se llama **implicación**.
 - \leftrightarrow se interpreta como “si y solo si”, se llama **doble implicación** o **equivalencia**.

9

Semántica clásica: tablas de verdad

Conociendo el valor de las variables proposicionales de una fórmula, podemos calcular el valor de verdad de la fórmula.

p	$\neg p$
V	F
F	V

p	q	$(p \wedge q)$
V	V	V
V	F	F
F	V	F
F	F	F

p	q	$(p \vee q)$
V	V	V
V	F	V
F	V	V
F	F	F

p	q	$(p \rightarrow q)$
V	V	V
V	F	F
F	V	V
F	F	V

p	q	$(p \leftrightarrow q)$
V	V	V
V	F	F
F	V	F
F	F	V

10

Ejemplo: tabla de verdad para $((p \wedge q) \rightarrow r)$

p	q	r	$(p \wedge q)$	$((p \wedge q) \rightarrow r)$
1	1	1	1	1
1	1	0	1	0
1	0	1	0	1
1	0	0	0	1
0	1	1	0	1
0	1	0	0	1
0	0	1	0	1
0	0	0	0	1

11

Tautologías, contradicciones y contingencias

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

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

p	q	$(p \wedge q)$	$((p \wedge q) \rightarrow p)$
V	V	V	V
V	F	F	V
F	V	F	V
F	F	F	V

- 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	$\neg p$	$(p \wedge \neg p)$
V	F	F
F	V	F

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

12

Equivalencias entre fórmulas

- Dos fórmulas A y B son **equivalentes** (y se escribe $A \equiv B$) si y sólo si, $A \leftrightarrow B$ es una tautología.

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

- | | |
|---|--|
| 1. Doble negación | $((p \wedge q) \leftrightarrow (q \wedge p))$ |
| $(\neg \neg p \leftrightarrow p)$ | $((p \vee q) \leftrightarrow (q \vee p))$ |
| 2. Idempotencia | 5. Distributividad |
| $((p \wedge p) \leftrightarrow p)$ | $((p \wedge (q \vee r)) \leftrightarrow ((p \wedge q) \vee (p \wedge r)))$ |
| $((p \vee p) \leftrightarrow p)$ | $((p \vee (q \wedge r)) \leftrightarrow ((p \vee q) \wedge (p \vee r)))$ |
| 3. Asociatividad | 6. Reglas de De Morgan |
| $((p \wedge q) \wedge r) \leftrightarrow (p \wedge (q \wedge r))$ | $(\neg(p \wedge q) \leftrightarrow (\neg p \vee \neg q))$ |
| $((p \vee q) \vee r) \leftrightarrow (p \vee (q \vee r))$ | $(\neg(p \vee q) \leftrightarrow (\neg p \wedge \neg q))$ |
| 4. Conmutatividad | |

13

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? Sí, False
7. ¿hay una fórmula más débil que todas? Sí, True

14

Expresión bien definida

- Toda expresión está **bien definida** 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 $x/y = 5$ 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 (V)
2. falso (F)
3. indefinido (\perp)

15

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	$(p \wedge_L q)$
V	V	V
V	F	F
F	V	F
F	F	F
V	\perp	\perp
F	\perp	F
\perp	V	\perp
\perp	F	\perp
\perp	\perp	\perp

p	q	$(p \vee_L q)$
V	V	V
V	F	V
F	V	V
F	F	F
V	\perp	V
F	\perp	\perp
\perp	V	\perp
\perp	F	\perp
\perp	\perp	\perp

16

Semántica trivaluada (secuencial)

¿Cuál es la tabla de verdad de \rightarrow_L ?

p	q	$(p \rightarrow_L q)$
V	V	V
V	F	F
F	V	V
F	F	V
V	\perp	\perp
F	\perp	V
\perp	V	\perp
\perp	F	\perp
\perp	\perp	\perp

17

Entonces...

Lógica proposicional y lógica trivaluada

- **Convención:** Dado que nuestros tipos de datos siempre tendrán como valor posible el indefinido o \perp , en general, asumiremos que estamos utilizando la lógica **trivaluada** por default.
- Es decir, salvo en los casos dónde se indique lo contrario:
 - \wedge podrá ser interpretado como \wedge_L directamente
 - y así con todos los operadores vistos.

18

Entonces... hablando de lógica proposicional

- ¿Es lo mismo decir...?
 - Mañana llueve e iré a comprar un paraguas
 - Si mañana llueve iré a comprar un paraguas
 - O mañana no llueve o no iré a comprar un paraguas
 - Compraré un paraguas por si mañana llueve
 - Si compro un paraguas, mañana llueve

19

Entonces... hablando de lógica proposicional

- Si llamamos:
 - a = Mañana llueve
 - b = Iré a comprar un paraguas
- Mañana llueve e iré a comprar un paraguas
Lo podríamos modelar como: $a \wedge b$
- Si mañana llueve iré a comprar un paraguas
Lo podríamos modelar como: $a \rightarrow b$
- O mañana no llueve o no iré a comprar un paraguas
Lo podríamos modelar como: $\neg a \vee \neg b$
- Compraré un paraguas por si mañana llueve
 - ¡A veces es difícil desambiguar!
 - Por si mañana llueve es una nueva proposición
- Si compro un paraguas, mañana llueve
Lo podríamos modelar como: $b \rightarrow a$

20

Práctica 1: Ejercicio 4

Determinar el valor de verdad de las siguientes proposiciones:

- a) $(\neg a \vee b)$
- b) $(c \vee (y \wedge x) \vee b)$

cuando el valor de verdad de a , b y c es *verdadero*, mientras que el de x e y es *falso*.

21

Práctica 1: Ejercicio 5

Determinar, utilizando tablas de verdad, si las siguientes fórmulas son tautologías, contradicciones o contingencias.

- b) $(p \wedge \neg p)$
- d) $((p \vee q) \rightarrow p)$
- i) $((p \wedge (q \vee r)) \leftrightarrow ((p \wedge q) \vee (p \wedge r)))$

22

Práctica 1: Ejercicio 6

Determinar la relación de fuerza de los siguientes pares de fórmulas:

- 1. *True*, *False* **False**

$$\alpha = (p \wedge q)$$

$$\beta = (p \vee q)$$

- 2. $(p \wedge q)$, $(p \vee q)$ **$(p \wedge q)$**

p	q	α	β	$\alpha \rightarrow \beta$	$\beta \rightarrow \alpha$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	1	1	1

- 3. *True*, *True* **True**

$$\alpha = (p \wedge q)$$

- 4. p , $(p \wedge q)$ **$(p \wedge q)$**

p	q	α	$\alpha \rightarrow p$	$p \rightarrow \alpha$
0	0	0	1	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	1

- 7. p , q **Ninguna es más fuerte**

23

Práctica 1: Ejercicio 7

Usando reglas de equivalencia (conmutatividad, asociatividad, De Morgan, etc) determinar si los siguientes pares de fórmulas son equivalencias. Indicar en cada paso qué regla se utilizó.

- 2. $\triangleright (p \vee q) \wedge (p \vee r)$
- $\triangleright (\neg p \rightarrow (q \wedge r))$

$$(\neg p \rightarrow (q \wedge r))$$

↓ Reemplazo implicación

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

↓ Distributiva

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

24

Práctica 1: Ejercicio 7

Usando reglas de equivalencia (conmutatividad, asociatividad, De Morgan, etc) determinar si los siguientes pares de fórmulas son equivalencias. Indicar en cada paso qué regla se utilizó.

6. ▶ $\neg(p \wedge (q \wedge s))$
▶ $(s \rightarrow (\neg p \vee \neg q))$

$$\begin{aligned} & \neg(p \wedge (q \wedge s)) \\ & \quad \downarrow \text{De Morgan} \\ & (\neg p \vee \neg(q \wedge s)) \\ & \quad \downarrow \text{De Morgan} \\ & (\neg p \vee \neg q \vee \neg s) \\ & \quad \downarrow \text{Conmutativa} \\ & (\neg s \vee \neg p \vee \neg q) \\ & \quad \downarrow \text{Reemplazo implicación} \\ & (s \rightarrow (\neg p \vee \neg q)) \end{aligned}$$

25

Práctica 1: Ejercicio 12

Sean las variables proposicionales f , e y m con los siguientes significados:

- ▶ $f \equiv$ "es fin de semana"
- ▶ $e \equiv$ "Juan estudia"
- ▶ $m \equiv$ "Juan escucha música"

Escribir usando lógica proposicional las siguientes oraciones:

1. "Si es fin de semana, Juan estudia o escucha música, pero no ambas cosas" $f \rightarrow ((e \vee m) \wedge \neg(e \wedge m))$
2. "Si no es fin de semana entonces Juan no estudia" $\neg f \rightarrow \neg e$
3. "Cuando Juan estudia los fines de semana, lo hace escuchando música" $(f \wedge e) \rightarrow m$

26

Práctica 1: Ejercicio 19

Determinar los valores de verdad de las siguientes proposiciones cuando el valor de verdad de b y c es *verdadero*, el de a es *falso* y el de x e y es *indefinido*:

- a) $(\neg x \vee_L b)$
- c) $\neg(c \vee y)$
- g) $(\neg c \wedge_L \neg y)$

27

Práctica 1: Ejercicio 20

Determinar los valores de las siguientes fórmulas de Lógica Ternaria cuando el valor de verdad de p es *verdadero*, el de q es *falso* y el de r es *indefinido*:

- a) $((9 \leq 9) \wedge p)$
- d) $((3 > 9) \vee (r \wedge (q \wedge p)))$
- i) $(p \wedge ((5 - 7 + 3 = 0)) \leftrightarrow (2^2 - 1 > 3))$

28

Práctica 1: Ejercicio 21

Sean p , q y r tres variables de las que se sabe que:

- ▶ p y q nunca están indefinidas,
- ▶ r se indefine sii q es verdadera

Proponer, para cada ítem, una fórmula que nunca se indefina, utilizando siempre las tres variables. Cada fórmula debe ser verdadera si y solo si se cumple que:

- b) Ninguna es verdadera.
- d) Sólo p y q son verdaderas.

29

Presentemos nuestro lenguaje de especificación

30

Problemas y Especificaciones

Inicialmente los problemas resolveremos con una computadora serán planteados como funciones. Es decir:

- ▶ Dados ciertos datos de entrada, obtendremos un resultado
- ▶ Más adelante en la materia, extenderemos el tipo de problemas que podemos resolver...

31

Definición (Especificación) de un problema

```
problema nombre(parámetros) : tipo de dato del resultado {  
  requiere etiqueta: { condiciones sobre los parámetros de entrada }  
  asegura etiqueta: { condiciones sobre los parámetros de salida }  
}
```

- ▶ *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:
 - ▶ Nombre del parámetro
 - ▶ Tipo de datos del parámetro
- ▶ *tipo de dato del resultado*: tipo de dato del resultado del problema (inicialmente especificaremos funciones)
 - ▶ En los asegura, podremos referenciar el valor devuelto con el nombre de **res**
- ▶ *etiquetas*: son nombres **opcionales** que nos servirán para nombrar declarativamente a las condiciones de los requiere o asegura.

32

Definición (Especificación) de un problema

► Sobre los *requiere*

- Describen todas las condiciones y posibles valores o casuísticas de los parámetros de entrada.
- Puede haber más de un *requiere* (recomendamos una condición por renglón). Se asume que valen todos juntos (es una conjunción).
- Evitar contradicciones (un *requiere* no debería contradecir a otro).

► Sobre los *asegura*

- Describen todas las condiciones y posibles valores o casuísticas de los parámetros de salida y entrada/salida en función de los parámetros de entrada.
- Puede haber más de un *asegura* (recomendamos una condición por renglón). Se asume que valen todos juntos (es una conjunción).
- Evitar contradicciones (un *asegura* no debería contradecir a otro).

33

¿Cómo contradicciones?

```
problema soyContradictorio(x :  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere esMayor: {x > 0}  
  requiere esMenor: {x < 0}  
  asegura esElSiguiente: {res + 1 = x}  
  asegura esElAnterior: {res - 1 = x}  
}
```

34

Ejemplos

```
problema raizCuadrada(x :  $\mathbb{R}$ ) :  $\mathbb{R}$  {  
  requiere: {x ≥ 0}  
  asegura: {res * res = x ∧ res ≥ 0}  
}
```

```
problema sumar(x :  $\mathbb{Z}$ , y :  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: {True}  
  asegura: {res = x + y}  
}
```

```
problema restar(x :  $\mathbb{Z}$ , y :  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: {True}  
  asegura: {res = x - y}  
}
```

```
problema cualquieramayor(x :  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: {True}  
  asegura: {res > x}  
}
```

35

¿Por qué nuestro lenguaje será semiformal?: Ejemplos

```
problema raizCuadrada(x :  $\mathbb{R}$ ) :  $\mathbb{R}$  {  
  requiere: {x debe ser mayor o igual que 0}  
  asegura: {res debe ser mayor o igual que 0}  
  asegura: {res elevado al cuadrado será x}  
}
```

```
problema sumar(x :  $\mathbb{Z}$ , y :  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: {-}  
  asegura: {res es la suma de x e y}  
}
```

```
problema restar(x :  $\mathbb{Z}$ , y :  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: {Siempre cumplen}  
  asegura: {res es la resta de x menos y}  
}
```

```
problema cualquieramayor(x :  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: {Vale para cualquier valor posible de x}  
  asegura: {res debe tener cualquier valor mayor a x}  
}
```

36

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?

37

Interpretando una especificación

- problema $\text{raizCuadrada}(x : \mathbb{R}) : \mathbb{R} \{$
 - requiere: $\{x \text{ debe ser mayor o igual que } 0\}$
 - asegura: $\{res \text{ debe ser mayor o igual que } 0\}$
 - asegura: $\{res \text{ elevado al cuadrado será } x\}$
 - $\}$
- ¿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 $res * res = x$ y $res \geq 0$.

38

Otro ejemplo

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

```
problema  $\text{cociente}(\text{dividendo} : \mathbb{Z}, \text{divisor} : \mathbb{Z}) : \mathbb{Z} \{$   
  requiere:  $\{\text{divisor} > 0\}$   
  asegura:  $\{res * \text{divisor} \leq \text{dividendo}\}$   
  asegura:  $\{(\text{res} + 1) * \text{divisor} > \text{dividendo}\}$   
}
```

Qué sucede si ejecutamos con ...

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

39

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

40

Tipos de datos de nuestro lenguaje de especificación

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

41

Tipo \mathbb{Z} (números enteros)

- ▶ Su **conjunto base** son los números enteros.
- ▶ Constantes: 0 ; 1 ; -1 ; 2 ; -2 ; ...
- ▶ Operaciones aritméticas:
 - ▶ $a + b$ (suma); $a - b$ (resta); $\text{abs}(a)$ (valor absoluto)
 - ▶ $a * b$ (multiplicación); $a \text{ div } b$ (división entera);
 - ▶ $a \bmod b$ (resto de dividir a por b), a^b o $\text{pot}(a,b)$ (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 $a \leq b$ (menor o igual)
 - ▶ $a > b$ (mayor)
 - ▶ $a \geq b$ o $a \geq b$ (mayor o igual)
 - ▶ $a = b$ (iguales)
 - ▶ $a \neq b$ (distintos)

42

Tipo \mathbb{R} (números reales)

- ▶ Su conjunto base son los números reales.
- ▶ Constantes: 0 ; 1 ; -7 ; 81 ; 7,4552 ; π ...
- ▶ 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 $a \leq b$ (menor o igual)
 - ▶ $a > b$ (mayor)
 - ▶ $a \geq b$ o $a \geq b$ (mayor o igual)
 - ▶ $a = b$ (iguales)
 - ▶ $a \neq b$ (distintos)

43

Tipo Bool (valor de verdad)

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

44

Tipo Char (caracteres)

- ▶ Sus elementos son las letras, dígitos y símbolos.
- ▶ Constantes: 'a', 'b', 'c', ..., 'z', ..., 'A', 'B', 'C', ..., 'Z', ..., '0', '1', '2', ..., '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 $\text{ord}(a) < \text{ord}(b)$.

45

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

46

Ejemplo de tipo enumerado

Definimos el tipo Día así:

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

Valen:

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

47