

Algoritmos y Estructuras de Datos I

Segundo cuatrimestre de 2025

Departamento de Computación - FCEyN - UBA

Lógica proposicional y Especificación de problemas

1

IP - AED I: Temario de la clase

- ▶ Lógica Proposicional
 - ▶ Motivación
 - ▶ Sintaxis
 - ▶ Semántica clásica
 - ▶ Tablas de verdad
 - ▶ Tautologías, contradicciones y contingencias
 - ▶ Noción de equivalencia de fórmulas y Relaciones de fuerza
 - ▶ Semántica trivaluada
 - ▶ Algunos ejercicios
- ▶ Presentación de nuestro lenguaje de especificación
 - ▶ Estructura de una especificación
 - ▶ Lenguaje semiformal
 - ▶ Contrato.
 - ▶ Interpretando especificaciones - Ejemplos
 - ▶ Tipos de datos
 - ▶ Básicos (\mathbb{Z} , \mathbb{R} , Bool, Char)
 - ▶ Tuplas (Uplas)
 - ▶ Secuencias
 - ▶ Renombre de tipos
 - ▶ Sobre-especificación y sub-especificación
 - ▶ Modularización en especificación

2

Se acuerdan 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

3

El abogado del diablo



- ▶ ¿Inocente o culpable?
 - ▶ Su torso está desnudo... pero... ¿y sus pies?
 - ▶ ¿Realmente estaba en el pasillo y en el ascensor al mismo tiempo?

4

Lógica proposicional

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

5

Lógica proposicional - Sintaxis

- ▶ Símbolos:

True , False , \neg , \wedge , \vee , \rightarrow , \leftrightarrow , (,)

- ▶ Variables proposicionales (infinitas)

p , q , r , ...

- ▶ Fórmulas

1. True y False son fórmulas
2. Cualquier variable proposicional (p , q , r , etc) 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

6

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

7

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**.

8

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

9

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

10

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.

11

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.
 - Doble negación
 $(\neg \neg p \leftrightarrow p)$
 - Idempotencia
 $((p \wedge p) \leftrightarrow p)$
 $((p \vee p) \leftrightarrow p)$
 - Asociatividad
 $((p \wedge q) \wedge r) \leftrightarrow (p \wedge (q \wedge r))$
 $((p \vee q) \vee r) \leftrightarrow (p \vee (q \vee r))$
 - Conmutatividad
 $((p \wedge q) \leftrightarrow (q \wedge p))$
 $((p \vee q) \leftrightarrow (q \vee p))$
 - Distributividad
 $((p \wedge (q \vee r)) \leftrightarrow ((p \wedge q) \vee (p \wedge r)))$
 $((p \vee (q \wedge r)) \leftrightarrow ((p \vee q) \wedge (p \vee r)))$
 - Reglas de De Morgan
 $(\neg(p \wedge q) \leftrightarrow (\neg p \vee \neg q))$
 $(\neg(p \vee q) \leftrightarrow (\neg p \wedge \neg q))$

12

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

13

Expresión bien definida

► Toda expresión está **bien definida** si todas las proposiciones valen V 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)

14

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

15

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

16

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.
- ▶ ¿Qué pasa con la definición de tautología, contradicción y contingencia en la lógica trivaluada?
 - ▶ Adaptaremos las definiciones para solo tener en cuenta los valores de verdad de las fórmulas cuando no están indefinidas
 - ▶ Por esto, cuando hacemos las tablas de verdad para analizar si una fórmula es tautología, contradicción o contingencia, pensaremos en la lógica bivaluada

17

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

18

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: $b \rightarrow a$

19

Práctica 1: Ejercicio 2

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*.

20

Práctica 1: Ejercicio 3

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

21

Práctica 1: Ejercicio 4

Determinar la relación de fuerza de los siguientes pares de fórmulas. Indicar cuál de las dos es más fuerte.:

- 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**

22

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

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

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.

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

$$\neg(p \wedge (q \wedge s))$$

↓ De Morgan

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

↓ De Morgan

$$(\neg p \vee \neg q \vee \neg s)$$

↓ Conmutativa

$$(\neg s \vee \neg p \vee \neg q)$$

↓ Reemplazo implicación

$$(s \rightarrow (\neg p \vee \neg q))$$

24

Presentemos nuestro Lenguaje de Especificación

25

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...

26

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.

27

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).

28

¿Cómo contradicciones?

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

29

Ejemplos

```
problema raizCuadrada( $x : \mathbb{R}$ ) :  $\mathbb{R}$  {  
  requiere:  $\{x \geq 0\}$   
  asegura:  $\{res \times res = x \wedge res \geq 0\}$   
}
```

```
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:  $\{res > x\}$   
}
```

30

¿Por qué nuestro lenguaje será semiformal?: Ejemplos

```
problema 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:  $\{res \text{ elevado al cuadrado será } x\}$   
}
```

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

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

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

31

El contrato

- **Contrato:** El programador escribe un programa P tal que si el usuario suministra datos que hacen verdadera la precondition, 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 precondition y la postcondición exactamente cuando se cumple el contrato.
- Si el usuario no cumple la precondition y P se cuelga o no cumple la poscondición...
 - ¿El usuario tiene derecho a quejarse?
 - ¿Se cumple el contrato?
- Si el usuario cumple la precondition y P se cuelga o no cumple la poscondición...
 - ¿El usuario tiene derecho a quejarse?
 - ¿Se cumple el contrato?

32

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 $res \times res = x$ y $res \geq 0$.

33

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:  $\{res \times \text{divisor} \leq \text{dividendo}\}$   
  asegura:  $\{(res + 1) \times \text{divisor} > \text{dividendo}\}$   
}
```

Qué sucede si ejecutamos con ...

- ▶ $\text{dividendo} = 1$ y $\text{divisor} = 0$?
- ▶ $\text{dividendo} = -4$ y $\text{divisor} = -2$, y obtenemos $res = 2$?
- ▶ $\text{dividendo} = -4$ y $\text{divisor} = -2$, y obtenemos $res = 0$?
- ▶ $\text{dividendo} = 4$ y $\text{divisor} = -2$, y el programa no termina?

34

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

35

Tipos de datos de nuestro lenguaje de especificación

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

36

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)
 - ▶ $a \times b$ (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)

37

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)

38

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 $a \neq b$)

39

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 $\text{ord}(a) < \text{ord}(b)$.

40

Tipo upla (o tupla)

- ▶ Una estructura de datos es una forma particular de organizar la información.
- ▶ 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.
- ▶ *n*ésimo: $(a_0, \dots, a_k)_m$ es el valor a_m en caso de que $0 \leq m \leq k$. Si no, está indefinido.
- ▶ Ejemplos:
 - ▶ $(7, 5)_0 = 7$
 - ▶ $(\text{'a'}, \text{true}, 78)_2 = 78$

41

Secuencias

- ▶ **Secuencia:** Varios elementos del mismo tipo T , posiblemente repetidos, ubicados en un cierto orden.
- ▶ $\text{seq}\langle T \rangle$ es el tipo de las secuencias cuyos elementos son de tipo T .
- ▶ T es un tipo arbitrario.
 - ▶ Hay secuencias de \mathbb{Z} , de Bool , de Días , de 5-uplas;
 - ▶ también hay secuencias de secuencias de T ;
 - ▶ etcétera.

42

Secuencias. Notación

- ▶ Una forma de escribir un elemento de tipo $\text{seq}\langle T \rangle$ es escribir términos de tipo T separados por comas, entre $\langle \dots \rangle$.
 - ▶ $\langle 1, 2, 3, 4, 1, 0 \rangle$ es una secuencia de \mathbb{Z} .
 - ▶ $\langle 1, 1 + 1, 3, 2 \times 2, 5 \bmod 2, 0 \rangle$ es otra secuencia de \mathbb{Z} (igual a la anterior).
- ▶ La **secuencia vacía** se escribe $\langle \rangle$, cualquiera sea el tipo de los elementos de la secuencia.
- ▶ Se puede formar secuencias de elementos de cualquier tipo.
 - ▶ Como $\text{seq}\langle \mathbb{Z} \rangle$ es un tipo, podemos armar secuencias de $\text{seq}\langle \mathbb{Z} \rangle$ (secuencias de secuencias de \mathbb{Z} , o sea $\text{seq}\langle \text{seq}\langle \mathbb{Z} \rangle \rangle$).
 - ▶ $\langle \langle 12, 13 \rangle, \langle -3, 9, 0 \rangle, \langle 5 \rangle, \langle \rangle, \langle \rangle, \langle 3 \rangle \rangle$ es un elemento de tipo $\text{seq}\langle \text{seq}\langle \mathbb{Z} \rangle \rangle$.

43

Secuencias bien formadas

Indicar si las siguientes secuencias están bien formadas. Si están bien formadas, indicar su tipo ($\text{seq}\langle \mathbb{Z} \rangle$, etc...)

- ▶ $\langle 1, 2, 3, 4, 5 \rangle$? Bien Formada. Tipa como $\text{seq}\langle \mathbb{Z} \rangle$ y $\text{seq}\langle \mathbb{R} \rangle$
- ▶ $\langle 1, \text{true}, 3, 4, 5 \rangle$? No está bien formada porque no es homogénea (Bool y \mathbb{Z})
- ▶ $\langle \text{'a'}, 2, 3, 4, 5 \rangle$? No está bien formada porque no es homogénea (Char y \mathbb{Z})
- ▶ $\langle \text{'H'}, \text{'o'}, \text{'l'}, \text{'a'} \rangle$? Bien Formada. Tipa como $\text{seq}\langle \text{Char} \rangle$
- ▶ $\langle \text{true}, \text{false}, \text{true}, \text{true} \rangle$? Bien Formada. Tipa como $\text{seq}\langle \text{Bool} \rangle$
- ▶ $\langle \frac{2}{5}, \pi, e \rangle$? Bien Formada. Tipa como $\text{seq}\langle \mathbb{R} \rangle$
- ▶ $\langle \rangle$? Bien formada. Tipa como cualquier secuencia $\text{seq}\langle X \rangle$ donde X es un tipo válido.
- ▶ $\langle \langle \rangle \rangle$? Bien formada. Tipa como cualquier secuencia $\text{seq}\langle \text{seq}\langle X \rangle \rangle$ donde X es un tipo válido.

44

Funciones sobre secuencias

Longitud

- ▶ Longitud: $length(a : seq\langle T \rangle) : \mathbb{Z}$
 - ▶ Representa la longitud de la secuencia a .
 - ▶ Notación: $length(a)$ se puede escribir como $|a|$ o como $a.length$.
- ▶ Ejemplos:
 - ▶ $|\langle \rangle| = 0$
 - ▶ $|\langle 'H', 'o', 'l', 'a' \rangle| = 4$
 - ▶ $|\langle 1, 1, 2 \rangle| = 3$

45

Funciones con secuencias

i-ésimo elemento

- ▶ Indexación: $seq\langle T \rangle[i : \mathbb{Z}] : T$
 - ▶ Requiere $0 \leq i < |a|$.
 - ▶ Es el elemento en la i -ésima posición de a .
 - ▶ La primera posición es la 0.
 - ▶ Notación: $a[i]$.
 - ▶ Si no vale $0 \leq i < |a|$ se indefine.
- ▶ Ejemplos:
 - ▶ $\langle 'H', 'o', 'l', 'a' \rangle[0] = 'H'$
 - ▶ $\langle 'H', 'o', 'l', 'a' \rangle[1] = 'o'$
 - ▶ $\langle 'H', 'o', 'l', 'a' \rangle[2] = 'l'$
 - ▶ $\langle 'H', 'o', 'l', 'a' \rangle[3] = 'a'$
 - ▶ $\langle 1, 1, 1, 1 \rangle[0] = 1$
 - ▶ $\langle \rangle[0] = \perp$ (Indefinido)
 - ▶ $\langle 1, 1, 1, 1 \rangle[7] = \perp$ (Indefinido)

46

Funciones con secuencias

Pertenece

- ▶ Pertenece: $pertenece(x : T, s : seq\langle T \rangle) : Bool$
 - ▶ Es **true** sí y solo sí x es elemento de s .
 - ▶ Notación: $pertenece(x, s)$ se puede escribir como $x \in s$.
- ▶ Ejemplos:
 - ▶ $(1, 'b') \in \langle (1, 'a'), (2, 'b'), (3, 'c'), (1, 'b') \rangle$? **true**
 - ▶ $(1, 'b') \in \langle (1, 'a'), (2, 'b'), (3, 'c'), (3, 'b') \rangle$? **false**

47

Funciones con secuencias

Igualdad

Dos secuencias s_0 y s_1 (notación $s_0 = s_1$) son iguales si y sólo si

- ▶ Tienen la misma cantidad de elementos
- ▶ Dada una posición, el elemento contenido en la secuencia s_0 es igual al elemento contenido en la secuencia s_1 .

Ejemplos:

- ▶ $\langle 1, 2, 3, 4 \rangle = \langle 1, 2, 3, 4 \rangle$? Sí
- ▶ $\langle \rangle = \langle \rangle$? Sí
- ▶ $\langle 4, 4, 4 \rangle = \langle 4, 4, 4 \rangle$? Sí
- ▶ $\langle 1, 2, 3, 4, 5 \rangle = \langle 1, 2, 3, 4 \rangle$? No
- ▶ $\langle 1, 2, 3, 4, 5 \rangle = \langle 1, 2, 4, 5, 6 \rangle$? No
- ▶ $\langle 1, 2, 3, 5, 4 \rangle = \langle 1, 2, 3, 4, 5 \rangle$? No

48

Funciones con secuencias

Cabeza o Head

- ▶ Cabeza: $\text{head}(a : \text{seq}(T)) : T$
 - ▶ Requiere $|a| > 0$.
 - ▶ Es el primer elemento de la secuencia a .
 - ▶ Es equivalente a la expresión $a[0]$.
 - ▶ Si no vale $|a| > 0$ se indefine.
- ▶ Ejemplos:
 - ▶ $\text{head}(\langle 'H', 'o', 'l', 'a' \rangle) = 'H'$
 - ▶ $\text{head}(\langle 1, 1, 1, 1 \rangle) = 1$
 - ▶ $\text{head}(\langle \rangle) = \perp$ (Indefinido)

49

Funciones con secuencias

Cola o Tail

- ▶ Cola: $\text{tail}(a : \text{seq}(T)) : \text{seq}(T)$
 - ▶ Requiere $|a| > 0$.
 - ▶ Es la secuencia resultante de eliminar su primer elemento.
 - ▶ Si no vale $|a| > 0$ se indefine.
- ▶ Ejemplos:
 - ▶ $\text{tail}(\langle 'H', 'o', 'l', 'a' \rangle) = \langle 'o', 'l', 'a' \rangle$
 - ▶ $\text{tail}(\langle 1, 1, 1, 1 \rangle) = \langle 1, 1, 1 \rangle$
 - ▶ $\text{tail}(\langle \rangle) = \perp$ (Indefinido)
 - ▶ $\text{tail}(\langle 6 \rangle) = \langle \rangle$

50

Funciones con secuencias

Agregar al principio o addFirst

- ▶ Agregar cabeza: $\text{addFirst}(t : T, a : \text{seq}(T)) : \text{seq}(T)$
 - ▶ Es una secuencia con los elementos de a , agregándole t como primer elemento.
 - ▶ Es una función que no se indefine
- ▶ Ejemplos:
 - ▶ $\text{addFirst}('x', \langle 'H', 'o', 'l', 'a' \rangle) = \langle 'x', 'H', 'o', 'l', 'a' \rangle$
 - ▶ $\text{addFirst}(5, \langle 1, 1, 1, 1 \rangle) = \langle 5, 1, 1, 1, 1 \rangle$
 - ▶ $\text{addFirst}(1, \langle \rangle) = \langle 1 \rangle$

51

Funciones con secuencias

Concatenación o concat

- ▶ Concatenación: $\text{concat}(a : \text{seq}(T), b : \text{seq}(T)) : \text{seq}(T)$
 - ▶ Es una secuencia con los elementos de a , seguidos de los de b .
 - ▶ Notación: $\text{concat}(a, b)$ se puede escribir $a ++ b$.
- ▶ Ejemplos:
 - ▶ $\text{concat}(\langle 'H', 'o' \rangle, \langle 'l', 'a' \rangle) = \langle 'H', 'o', 'l', 'a' \rangle$
 - ▶ $\text{concat}(\langle 1, 2 \rangle, \langle 3, 4 \rangle) = \langle 1, 2, 3, 4 \rangle$
 - ▶ $\text{concat}(\langle \rangle, \langle \rangle) = \langle \rangle$
 - ▶ $\text{concat}(\langle 2, 3 \rangle, \langle \rangle) = \langle 2, 3 \rangle$
 - ▶ $\text{concat}(\langle \rangle, \langle 5, 7 \rangle) = \langle 5, 7 \rangle$

52

Funciones con secuencias

Subsecuencia o subseq

- ▶ Subsecuencia: $\text{subseq}(a : \text{seq}\langle T \rangle, d, h : \mathbb{Z}) : \text{seq}\langle T \rangle$
 - ▶ Es una sublista de a en las posiciones entre d (inclusive) y h (exclusive).
 - ▶ Cuando $0 \leq d = h \leq |a|$, retorna la secuencia vacía.
 - ▶ Cuando no se cumple $0 \leq d \leq h \leq |a|$, se **indefine!**
- ▶ Ejemplos:
 - ▶ $\text{subseq}(\langle 'H', 'o', 'l', 'a' \rangle, 0, 1) = \langle 'H' \rangle$
 - ▶ $\text{subseq}(\langle 'H', 'o', 'l', 'a' \rangle, 0, 4) = \langle 'H', 'o', 'l', 'a' \rangle$
 - ▶ $\text{subseq}(\langle 'H', 'o', 'l', 'a' \rangle, 2, 2) = \langle \rangle$
 - ▶ $\text{subseq}(\langle 'H', 'o', 'l', 'a' \rangle, -1, 3) = \perp$
 - ▶ $\text{subseq}(\langle 'H', 'o', 'l', 'a' \rangle, 0, 10) = \perp$
 - ▶ $\text{subseq}(\langle 'H', 'o', 'l', 'a' \rangle, 3, 1) = \perp$

53

Funciones con secuencias

Modificar un valor o setAt

- ▶ Cambiar una posición: $\text{setAt}(a : \text{seq}\langle T \rangle, i : \mathbb{Z}, val : T) : \text{seq}\langle T \rangle$
 - ▶ Requiere $0 \leq i < |a|$
 - ▶ Es una secuencia igual a a , pero con valor val en la posición i .
- ▶ Ejemplos:
 - ▶ $\text{setAt}(\langle 'H', 'o', 'l', 'a' \rangle, 0, 'X') = \langle 'X', 'o', 'l', 'a' \rangle$
 - ▶ $\text{setAt}(\langle 'H', 'o', 'l', 'a' \rangle, 3, 'A') = \langle 'H', 'o', 'l', 'A' \rangle$
 - ▶ $\text{setAt}(\langle \rangle, 0, 5) = \perp$ (Indefinido)

54

Operaciones sobre secuencias

- ▶ $\text{length}(a : \text{seq}\langle T \rangle) : \mathbb{Z}$ (notación $|a|$)
- ▶ $\text{pertenece}(x : T, s : \text{seq}\langle T \rangle) : \text{Bool}$ (notación $x \in s$)
- ▶ indexación: $\text{seq}\langle T \rangle[i : \mathbb{Z}] : T$
- ▶ igualdad: $\text{seq}\langle T \rangle = \text{seq}\langle T \rangle$
- ▶ $\text{head}(a : \text{seq}\langle T \rangle) : T$
- ▶ $\text{tail}(a : \text{seq}\langle T \rangle) : \text{seq}\langle T \rangle$
- ▶ $\text{addFirst}(t : T, a : \text{seq}\langle T \rangle) : \text{seq}\langle T \rangle$
- ▶ $\text{concat}(a : \text{seq}\langle T \rangle, b : \text{seq}\langle T \rangle) : \text{seq}\langle T \rangle$ (notación $a++b$)
- ▶ $\text{subseq}(a : \text{seq}\langle T \rangle, d, h : \mathbb{Z}) : \langle T \rangle$
- ▶ $\text{setAt}(a : \text{seq}\langle T \rangle, i : \mathbb{Z}, val : T) : \text{seq}\langle T \rangle$

55

Renombre de tipos

- ▶ Un renombre de tipos (o *alias* en inglés) en un lenguaje es una forma de crear un nuevo nombre para un tipo de dato que ya existe.
- ▶ Este nuevo nombre no crea un nuevo tipo de dato, sino que simplemente actúa como un sinónimo del tipo original.
- ▶ Puede ser útil para hacer la especificación más legible o para adaptar un tipo genérico a un contexto específico.
- ▶ En nuestro lenguaje de especificación vamos a adoptar la notación $\text{Renombre } T_1 = T_2$
- ▶ Ejemplos:
 - ▶ Un usuario en un sistema puede estar representado con una tupla de 2 elementos, donde el primero corresponde al número de identificación (id) y el segundo a su nombre de usuario. Entonces:
 $\text{Renombre Usuario} = \mathbb{Z} \times \text{seq}\langle \text{Char} \rangle$
 - ▶ Una publicación de un usuario en una red social puede representarse con una tupla de 3 elementos compuesta por: el autor de dicha publicación, el texto publicado y el conjunto de los usuarios que le dieron *me gusta*. Entonces:
 $\text{Renombre Publicación} = \text{Usuario} \times \text{seq}\langle \text{Char} \rangle \times \text{seq}\langle \text{Usuario} \rangle$

56

Problemas comunes de las especificaciones

- ▶ ¿Qué sucede si especifico de menos?
- ▶ ¿Qué sucede si especifico de más?

57

Sobre-especificación

- ▶ Consiste en dar una **postcondición más restrictiva** de la 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:

```
problema distinto(x :  $\mathbb{Z}$ ) :  $\mathbb{Z}$  {  
  requiere: { True }  
  asegura: { res = x + 1 }  
}
```
- ▶ ... en lugar de:

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

58

Sub-especificación

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

```
problema distinto(x :  $\mathbb{Z}$ ) :  $\mathbb{Z}$ {  
  requiere: { x > 0 }  
  asegura: { res  $\neq$  x }  
}
```

... en vez de:

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

59

Modularización

Partiendo un problema en problemas mas chicos

Dadas dos secuencias, queremos saber si uno es una permutación¹ de la otra secuencia:

¿Cuándo será una secuencia permutación de la otra?

- ▶ Tienen los mismos elementos
- ▶ Cada elemento aparece la misma cantidad de veces en ambas secuencias

```
problema esPermutacion(s1, s2 : seq( $\mathcal{T}$ )) : Bool {  
  asegura: { res = true  $\leftrightarrow$  para cada elemento es cierto que tiene la misma  
    cantidad de apariciones en s1 y s2 }  
}
```

Pero... falta algo...

¹mismos elementos y misma cantidad por cada elemento, en un orden potencialmente distinto

60

Modularización

Partiendo un problema en problemas mas chicos

Ahora, tenemos que especificar el problema *cantidadDeApariciones*

¿Cómo podemos saber la cantidad de apariciones de un elemento en una lista?

- ▶ Podríamos sumar 1 por cada posición donde el elemento en dicha posición es el que buscamos!
- ▶ Las operaciones de Sumatorias y Productorias también podemos usarlos

```
problema cantidadDeApariciones(s : seq<T>, e : T) : ℤ {  
  asegura {res = la cantidad de veces que el elemento e aparece en la lista s }  
}
```

61

Recapitulando

Partiendo un problema en problemas mas chicos

Dadas dos secuencias, queremos saber si uno es una permutación¹ de la otra secuencia:

```
problema esPermutacion(s1, s2 : seq<T>) : Bool {  
  asegura: {res = true ↔ (para cada elemento e de T, se cumple que  
    (cantidadDeApariciones(s1, e) = cantidadDeApariciones(s2, e)))}  
}
```

Donde...

```
problema cantidadDeApariciones(s : seq<T>, e : T) : ℤ {  
  asegura {res = la cantidad de veces que el elemento e aparece en la lista s }  
}
```

Y así podemos modularizar y descomponer nuestro problemas, partiendolos en problemas más chicos. Y también los podremos reutilizar!

¹mismos elementos y misma cantidad por cada elemento, en un orden potencialmente distinto

62

Modularización

O partir el problema en problemas más chicos...

Los conceptos de modularización y encapsulamiento siempre estarán relacionados con los principios de diseño de software. La estrategia se puede resumir en:

- ▶ Descomponer un problema grande en problemas más pequeños (y sencillos)
- ▶ Componerlos y obtener la solución al problema original

Esto favorece muchos aspectos de calidad como:

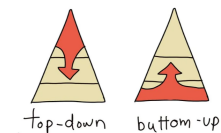
- ▶ La reutilización (una función auxiliar puede ser utilizada en muchos contextos)
- ▶ Es más facil probar algo chico que algo grande (si cada parte cumple su función correctamente, es más probable que todas juntas también lo haga)
- ▶ La declaratividad (es más facil entender al ojo humano)

63

Modularización

Top Down versus Bottom Up

También es aplicable a la especificación de problemas:



```
problema esPermutacion(s1, s2 : seq<T>) : Bool {  
  asegura: {res = true ↔ (para cada elemento e de T, se cumple que  
    (cantidadDeApariciones(s1, e) = cantidadDeApariciones(s2, e)))}  
}
```

```
problema cantidadDeApariciones(s : seq<T>, e : T) : ℤ {  
  asegura {res = la cantidad de veces que el elemento e aparece en la lista s }  
}
```

¿Lo encaramos Top Down o Bottom Up?

64