Introducción a la Programación

Algoritmos y Estructuras de Datos I

Segundo cuatrimestre de 2025

Lógica proposicional y Especificación de problemas

IP - AED I: Temario de la clase

- Lógica Proposicional
  - Motivación
  - Sintáxis
  - 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

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 paragüas
  - Si mañana llueve iré a comprar un paragüas
  - O mañana no llueve o no iré a comprar un paragüas
  - Compraré un paragüas por si mañana llueve
  - Si compro un paragüas, mañana llueve

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?

Lógica proposicional

- Es la lógica que habla sobre las proposiciones.

Lógica proposicional

- Es la lógica que habla sobre las proposiciones.
- Son oraciones que tienen un valor de verdad, Verdadero o Falso .

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.

Lógica proposicional - Sintaxis

- Símbolos:

$$\text{True}\ ,\ \text{False}\ ,\ \neg\ ,\ \wedge\ ,\ \vee\ ,\ \rightarrow\ ,\ \leftrightarrow\ ,\ (\ ,\ )$$

Lógica proposicional - Sintaxis

- Símbolos:

$$\text{True} \ , \ \text{False} \ , \ \neg \ , \ \wedge \ , \ \vee \ , \ \rightarrow \ , \ \leftrightarrow \ , \ (\ , \ )$$

- Variables proposicionales (infinitas)

$$p \ , \ q \ , \ r \ , \ \ldots$$

Lógica proposicional - Sintaxis

- Símbolos:

$$\text{True} \ , \ \text{False} \ , \ \neg \ , \ \wedge \ , \ \vee \ , \ \rightarrow \ , \ \leftrightarrow \ , \ (\ , \ )$$

- Variables proposicionales (infinitas)

$$p \ , \ q \ , \ r \ , \ \ldots$$

- Fórmulas
  1. True y False son fórmulas

Lógica proposicional - Sintaxis

- Símbolos:

$$\text{True} \ , \ \text{False} \ , \ \neg \ , \ \wedge \ , \ \vee \ , \ \rightarrow \ , \ \leftrightarrow \ , \ (\ , \ )$$

- Variables proposicionales (infinitas)

$$p \ , \ q \ , \ r \ , \ \ldots$$

- Fórmulas

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

Lógica proposicional - Sintaxis

- Símbolos:

$$\text{True} \ , \ \text{False} \ , \ \neg \ , \ \wedge \ , \ \vee \ , \ \rightarrow \ , \ \leftrightarrow \ , \ (\ , \ )$$

- Variables proposicionales (infinitas)

$$p \ , \ q \ , \ r \ , \ \ldots$$

- 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

Lógica proposicional - Sintaxis

- Símbolos:

$$\text{True} \ , \ \text{False} \ , \ \neg \ , \ \wedge \ , \ \vee \ , \ \rightarrow \ , \ \leftrightarrow \ , \ (\ , \ )$$

- Variables proposicionales (infinitas)

$$p \ , \ q \ , \ r \ , \ \ldots$$

- 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 $A_1, A_2, \ldots, A_n$ son fórmulas, $(A_1 \wedge A_2 \wedge \cdots \wedge A_n)$ es una fórmula

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 (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, $(A_1 \wedge A_2 \wedge \cdots \wedge A_n)$ es una fórmula
5. Si $A_1, A_2, \dots, A_n$ son fórmulas, $(A_1 \vee A_2 \vee \cdots \vee A_n)$ es una fórmula

Lógica proposicional - Sintaxis

- Símbolos:

$$\text{True} \ , \ \text{False} \ , \ \neg \ , \ \wedge \ , \ \vee \ , \ \rightarrow \ , \ \leftrightarrow \ , \ (\ , \ )$$

- Variables proposicionales (infinitas)

$$p \ , \ q \ , \ r \ , \ \ldots$$

- 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 $A_1, A_2, \ldots, A_n$ son fórmulas, $(A_1 \wedge A_2 \wedge \cdots \wedge A_n)$ es una fórmula
  5. Si $A_1, A_2, \ldots, A_n$ son fórmulas, $(A_1 \vee A_2 \vee \cdots \vee A_n)$ es una fórmula
  6. Si A y B son fórmulas, $(A \rightarrow B)$ es una fórmula

Lógica proposicional - Sintaxis

- Símbolos:

$$\text{True} \ , \ \text{False} \ , \ \neg \ , \ \wedge \ , \ \vee \ , \ \rightarrow \ , \ \leftrightarrow \ , \ (\ , \ )$$

- Variables proposicionales (infinitas)

$$p \ , \ q \ , \ r \ , \ \ldots$$

- 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 $A_1, A_2, \ldots, A_n$ son fórmulas, $(A_1 \wedge A_2 \wedge \cdots \wedge A_n)$ es una fórmula
  5. Si $A_1, A_2, \ldots, A_n$ son fórmulas, $(A_1 \vee A_2 \vee \cdots \vee A_n)$ 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

Ejemplos

¿Cuáles son fórmulas?

- $p \vee q$
- $(p \vee q)$
- $p \vee q \rightarrow r$
- $(p \vee q) \rightarrow r$
- $((p \vee q) \rightarrow r)$
- $(p \rightarrow q \rightarrow r)$

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

Semántica clásica

- Dos valores de verdad: “verdadero” (V) y “falso” (F).

Semántica clásica

- Dos valores de verdad: "verdadero" (V) y "falso" (F).
- Interpretación:
  - True siempre vale V.

Semántica clásica

- Dos valores de verdad: “verdadero” (V) y “falso” (F).
- Interpretación:
  - True siempre vale V.
  - False siempre vale F.

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.

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.

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.

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.

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.

Semántica clásica: tablas de verdad

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

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	

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	

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

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	F	
F	V	
F	F	

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	V	
F	F	

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	

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	

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

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	F	
F	V	
F	F	

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	
F	V	
F	F	

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	
F	F	

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	

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

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

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

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	
F	F	

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	

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

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	F	
F	V	
F	F	

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	V	
F	F	

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	$(p \to q)$
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	

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	

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

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	0		
1	0	1		
1	0	0		
0	1	1		
0	1	0		
0	0	1		
0	0	0		

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	0	1	
1	0	1	0	
1	0	0	0	
0	1	1	0	
0	1	0	0	
0	0	1	0	
0	0	0	0	

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

Tautologías, contradicciones y contingencias

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

Tautologías, contradicciones y contingencias

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

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

Tautologías, contradicciones y contingencias

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

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

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

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

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.

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	V	

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

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.

Equivalencias entre fórmulas

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

Equivalencias entre fórmulas

- Dos fórmulas A y es B son equivalentes (y se escribe $A \equiv B$) si y sólo si, $A \leftrightarrow B$ es una tautologia.
- **Teorema:** Las siguientes fórmulas son tautologías.

Equivalencias entre fórmulas

- Dos fórmulas A y es B son equivalentes (y se escribe $A \equiv B$) si y sólo si, $A \leftrightarrow B$ es una tautologia.
- **Teorema:** Las siguientes fórmulas son tautologías.
  1. Doble negación
     $(\neg\neg p \leftrightarrow p)$

Equivalencias entre fórmulas

- Dos fórmulas A y es B son equivalentes (y se escribe $A \equiv B$) si y sólo si, $A \leftrightarrow B$ es una tautologia.
- **Teorema:** Las siguientes fórmulas son tautologías.
  1. Doble negación
     $(\neg\neg p \leftrightarrow p)$
  2. Idempotencia
     $((p \wedge p) \leftrightarrow p)$
     $((p \vee p) \leftrightarrow p)$

Equivalencias entre fórmulas

- Dos fórmulas A y es B son equivalentes (y se escribe $A \equiv B$) si y sólo si, $A \leftrightarrow B$ es una tautologia.
- **Teorema:** Las siguientes fórmulas son tautologías.
  1. Doble negación
     $(\neg\neg p \leftrightarrow p)$
  2. Idempotencia
     $((p \wedge p) \leftrightarrow p)$
     $((p \vee p) \leftrightarrow p)$
  3. Asociatividad
     $(((p \wedge q) \wedge r) \leftrightarrow (p \wedge (q \wedge r)))$
     $(((p \vee q) \vee r) \leftrightarrow (p \vee (q \vee r)))$

Equivalencias entre fórmulas

- Dos fórmulas A y es B son equivalentes (y se escribe $A \equiv B$) si y sólo si, $A \leftrightarrow B$ es una tautologia.
- **Teorema:** Las siguientes fórmulas son tautologías.

  1. Doble negación
     $(\neg\neg p \leftrightarrow p)$
  2. Idempotencia
     $((p \wedge p) \leftrightarrow p)$
     $((p \vee p) \leftrightarrow p)$
  3. Asociatividad
     $(((p \wedge q) \wedge r) \leftrightarrow (p \wedge (q \wedge r)))$
     $(((p \vee q) \vee r) \leftrightarrow (p \vee (q \vee r)))$
  4. Conmutatividad
     $((p \wedge q) \leftrightarrow (q \wedge p))$ $((p \vee q) \leftrightarrow (q \vee p))$

Equivalencias entre fórmulas

- Dos fórmulas A y es B son equivalentes (y se escribe $A \equiv B$) si y sólo si, $A \leftrightarrow B$ es una tautologia.
- **Teorema:** Las siguientes fórmulas son tautologías.
  1. Doble negación
     $(\neg\neg p \leftrightarrow p)$
  2. Idempotencia
     $((p \wedge p) \leftrightarrow p)$
     $((p \vee p) \leftrightarrow p)$
  3. Asociatividad
     $(((p \wedge q) \wedge r) \leftrightarrow (p \wedge (q \wedge r)))$
     $(((p \vee q) \vee r) \leftrightarrow (p \vee (q \vee r)))$
  4. Conmutatividad
     $((p \wedge q) \leftrightarrow (q \wedge p))$ $((p \vee q) \leftrightarrow (q \vee p))$
  5. Distributividad
     $((p \wedge (q \vee r)) \leftrightarrow ((p \wedge q) \vee (p \wedge r)))$
     $((p \vee (q \wedge r)) \leftrightarrow ((p \vee q) \wedge (p \vee r)))$

Equivalencias entre fórmulas

- Dos fórmulas A y es B son equivalentes (y se escribe $A \equiv B$) si y sólo si, $A \leftrightarrow B$ es una tautologia.
- **Teorema:** Las siguientes fórmulas son tautologías.

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

Relación de fuerza

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

Relación de fuerza

- Decimos que A es más fuerte que B cuando $(A \to B)$ es tautología.
- También decimos que A fuerza a B o que B es más débil que A.

Relación de fuerza

- Decimos que A es más fuerte que B cuando $(A \to B)$ 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?

Relación de fuerza

- Decimos que A es más fuerte que B cuando $(A \to B)$ 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

Relación de fuerza

- Decimos que A es más fuerte que B cuando $(A \to B)$ 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?

Relación de fuerza

- Decimos que A es más fuerte que B cuando $(A \to B)$ 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

Relación de fuerza

- Decimos que A es más fuerte que B cuando $(A \to B)$ 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 $(q \to p)$?

Relación de fuerza

- Decimos que A es más fuerte que B cuando $(A \to B)$ 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 $(q \to p)$? Sí
     Pero notemos que si q está indefinido y p es verdadero entonces $(q \to p)$ está indefinido.
  4. ¿p es más fuerte que q?

Relación de fuerza

- Decimos que A es más fuerte que B cuando $(A \to B)$ 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 $(q \to p)$? Sí
     Pero notemos que si q está indefinido y p es verdadero entonces $(q \to p)$ está indefinido.
  4. ¿p es más fuerte que q? No

Relación de fuerza

- Decimos que A es más fuerte que B cuando $(A \to B)$ 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 $(q \to p)$? Sí
     Pero notemos que si q está indefinido y p es verdadero entonces $(q \to p)$ está indefinido.
  4. ¿p es más fuerte que q? No
  5. ¿p es más fuerte que p?

Relación de fuerza

- Decimos que A es más fuerte que B cuando $(A \to B)$ 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 $(q \to p)$? Sí
     Pero notemos que si q está indefinido y p es verdadero entonces $(q \to p)$ está indefinido.
  4. ¿p es más fuerte que q? No
  5. ¿p es más fuerte que p? Sí

Relación de fuerza

- Decimos que A es más fuerte que B cuando $(A \to B)$ 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 $(q \to p)$? Sí
     Pero notemos que si q está indefinido y p es verdadero entonces $(q \to p)$ 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?

Relación de fuerza

- Decimos que A es más fuerte que B cuando $(A \to B)$ 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 $(q \to p)$? Sí
     Pero notemos que si q está indefinido y p es verdadero entonces $(q \to p)$ 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

Relación de fuerza

- Decimos que A es más fuerte que B cuando $(A \to B)$ 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 $(q \to p)$? Sí
     Pero notemos que si q está indefinido y p es verdadero entonces $(q \to p)$ 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?

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

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 ($\perp$)

Semántica trivaluada (secuencial)

Se llama secuencial porque ...

Semántica trivaluada (secuencial)

Se llama secuencial porque ...

- los términos se evalúan de izquierda a derecha,

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.

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**).

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	$\perp$	
F	$\perp$	
$\perp$	V	
$\perp$	F	
$\perp$	$\perp$	

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	$\perp$	
F	$\perp$	
$\perp$	V	
$\perp$	F	
$\perp$	$\perp$	

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

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	$\perp$	
$\perp$	V	
$\perp$	F	
$\perp$	$\perp$	

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

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	
$\perp$	F	
$\perp$	$\perp$	

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

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	$\perp$	
F	$\perp$	
$\perp$	V	
$\perp$	F	
$\perp$	$\perp$	

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	$\perp$	
$\perp$	V	
$\perp$	F	
$\perp$	$\perp$	

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	
$\perp$	F	
$\perp$	$\perp$	

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	$\perp$
$\perp$	$\perp$	$\perp$

Semántica trivaluada (secuencial)

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

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

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	$\perp$	
$\perp$	V	
$\perp$	F	
$\perp$	$\perp$	

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	
$\perp$	F	
$\perp$	$\perp$	

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	$\perp$
$\perp$	$\perp$	$\perp$

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.

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.

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

Entonces... hablando de lógica proposicional

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

Entonces... hablando de lógica proposicional

- Si llamamos:
  - a = Mañana llueve
  - b = Iré a comprar un paragüas
- Mañana llueve e iré a comprar un paragüas

Entonces... hablando de lógica proposicional

- Si llamamos:
  - $a =$ Mañana llueve
  - $b =$ Iré a comprar un paragüas

- Mañana llueve e iré a comprar un paragüas
  Lo podriamos modelar como: $a \wedge b$

Entonces... hablando de lógica proposicional

- Si llamamos:
  - a = Mañana llueve
  - b = Iré a comprar un paragüas

- Mañana llueve e iré a comprar un paragüas
  Lo podriamos modelar como: $a \wedge b$
- Si mañana llueve iré a comprar un paragüas

Entonces... hablando de lógica proposicional

- Si llamamos:
  - a = Mañana llueve
  - b = Iré a comprar un paragüas
- Mañana llueve e iré a comprar un paragüas
  Lo podriamos modelar como: $a \wedge b$
- Si mañana llueve iré a comprar un paragüas
  Lo podriamos modelar como: $a \rightarrow b$

Entonces... hablando de lógica proposicional

- Si llamamos:
  - a = Mañana llueve
  - b = Iré a comprar un paragüas

- Mañana llueve e iré a comprar un paragüas
  Lo podriamos modelar como: $a \wedge b$
- Si mañana llueve iré a comprar un paragüas
  Lo podriamos modelar como: $a \rightarrow b$
- O mañana no llueve o no iré a comprar un paragüas

Entonces... hablando de lógica proposicional

- Si llamamos:
  - a = Mañana llueve
  - b = Iré a comprar un paragüas

- Mañana llueve e iré a comprar un paragüas
  Lo podriamos modelar como: $a \wedge b$
- Si mañana llueve iré a comprar un paragüas
  Lo podriamos modelar como: $a \rightarrow b$
- O mañana no llueve o no iré a comprar un paragüas
  Lo podriamos modelar como: $\neg a \vee \neg b$

Entonces... hablando de lógica proposicional

- Si llamamos:
  - a = Mañana llueve
  - b = Iré a comprar un paragüas

- Mañana llueve e iré a comprar un paragüas
  Lo podriamos modelar como: $a \wedge b$
- Si mañana llueve iré a comprar un paragüas
  Lo podriamos modelar como: $a \rightarrow b$
- O mañana no llueve o no iré a comprar un paragüas
  Lo podriamos modelar como: $\neg a \vee \neg b$
- Compraré un paragüas por si mañana llueve

Entonces... hablando de lógica proposicional

- Si llamamos:
  - $a =$ Mañana llueve
  - $b =$ Iré a comprar un paragüas

- Mañana llueve e iré a comprar un paragüas
  Lo podriamos modelar como: $a \land b$
- Si mañana llueve iré a comprar un paragüas
  Lo podriamos modelar como: $a \rightarrow b$
- O mañana no llueve o no iré a comprar un paragüas
  Lo podriamos modelar como: $\neg a \lor \neg b$
- Compraré un paragüas por si mañana llueve
  - ¡A veces es difícil desambigüar!
  - Por si mañana llueve es una nueva proposición

Entonces... hablando de lógica proposicional

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

Entonces... hablando de lógica proposicional

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

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*.

Práctica 1: Ejercicio 3

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

b) $(p \wedge \neg p)$

d) $((p \vee q) \to p)$

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

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*

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*

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

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

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

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

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

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

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

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*
2. $(p \land q)$, $(p \lor q)$ $(p \land q)$
3. *True*, *True*

Práctica 1: Ejercicio 4

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

1. $True$, $False$ $False$
2. $(p \wedge q)$, $(p \vee q)$ $(p \wedge q)$
3. $True$, $True$ $True$

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*
2. $(p \wedge q)$, $(p \vee q)$ $(p \wedge q)$
3. *True*, *True* *True*
4. p, $(p \wedge q)$

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*
2. $(p \wedge q)$, $(p \vee q)$ $(p \wedge q)$
3. *True*, *True* *True*
4. p, $(p \wedge q)$

$\alpha = (p \wedge q)$

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

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*
2. $(p \wedge q)$, $(p \vee q)$ $(p \wedge q)$
3. *True*, *True* *True*
4. p, $(p \wedge q)$ $(p \wedge q)$

$\alpha = (p \wedge q)$

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

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*
2. $(p \wedge q)$, $(p \vee q)$ $(p \wedge q)$
3. *True*, *True* *True*
4. p, $(p \wedge q)$ $(p \wedge q)$

7. p, q

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*
2. $(p \wedge q)$, $(p \vee q)$ $(p \wedge q)$
3. *True*, *True* *True*
4. p, $(p \wedge q)$ $(p \wedge q)$

7. p, q Ninguna es más fuerte

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

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. - $(p \lor q) \land (p \lor r)$
   - $(\neg p \to (q \land r))$

$$(\neg p \to (q \land r))$$

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

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

$\downarrow$ Reemplazo implicación

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

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

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

$$\downarrow \text{Reemplazo implicación}$$

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

$$\downarrow \text{Distributiva}$$

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

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

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

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

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

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

$$\downarrow \text{De Morgan}$$

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

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

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

$$\downarrow \text{De Morgan}$$

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

$$\downarrow \text{De Morgan}$$

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

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

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

$$\downarrow \text{De Morgan}$$

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

$$\downarrow \text{De Morgan}$$

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

$$\downarrow \text{Conmutativa}$$

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

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

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

Presentemos nuestro Lenguaje de Especificación

Problemas y Especificaciones (contratos)

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...

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 aseguras.

Definición (Especificación) de un problema

- *Sobre los requiere*
  - Describen todas las condiciones y posibles valores o casuísticas de los parámetros de entrada (precondiciones).
  - 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 (postcondiciones).
  - 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).

¿Cómo contradicciones?

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

Ejemplos

problema *raizCuadrada*$(x : \mathbb{R}) : \mathbb{R}$ {

Ejemplos

problema $raizCuadrada(x : \mathbb{R}) : \mathbb{R}$ {
requiere: $\{x \geq 0\}$

Ejemplos

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

Ejemplos

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

problema $sumar(x : \mathbb{Z}, y : \mathbb{Z}) : \mathbb{Z}$ {

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: $\{True\}$

Ejemplos

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

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

Ejemplos

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

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

problema *restar*$(x : \mathbb{Z}, y : \mathbb{Z}) : \mathbb{Z}$ {

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: $\{True\}$

Ejemplos

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

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: $\{res = x - y\}$
}

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 $cualquieramayor(x : \mathbb{Z}) : \mathbb{Z}$ {

Ejemplos

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

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: $\{True\}$

Ejemplos

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

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

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

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

¿Por qué nuestro lenguaje será semiformal?: Ejemplos

problema *raizCuadrada*$(x : \mathbb{R}) : \mathbb{R}$ {

¿Por qué nuestro lenguaje será semiformal?: Ejemplos

problema *raizCuadrada*($x : \mathbb{R}$) : $\mathbb{R}$ {
  requiere: {x debe ser mayor o igual que 0}

¿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}

¿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}
}

¿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*$(x : \mathbb{Z}, y : \mathbb{Z}) : \mathbb{Z}$ {

¿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: {$-$}

¿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 $y\}$
}

¿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*$(x : \mathbb{Z}, y : \mathbb{Z}) : \mathbb{Z}$ $\{$

¿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}

¿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}
}

¿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*$(x : \mathbb{Z}) : \mathbb{Z}$ {

¿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}

¿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}
}

El contrato

- **Contrato:** *El programador escribe un programa P tal que si el usuario suministra datos que hacen verdadera la precondición (cláusulas requiere), entonces P termina en una cantidad finita de pasos retornando un valor que hace verdadera la postcondición (cláusulas asegura).*

El contrato

- **Contrato:** *El programador escribe un programa* P *tal que si el usuario suministra datos que hacen verdadera la precondición (cláusulas requiere), entonces* P *termina en una cantidad finita de pasos retornando un valor que hace verdadera la postcondición (cláusulas asegura).*
- El programa P es correcto para la especificación dada por la precondición y la postcondición exactamente cuando se cumple el contrato.

El contrato

- **Contrato:** *El programador escribe un programa P tal que si el usuario suministra datos que hacen verdadera la precondición (cláusulas requiere), entonces P termina en una cantidad finita de pasos retornando un valor que hace verdadera la postcondición (cláusulas asegura).*
- 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?

El contrato

- **Contrato:** *El programador escribe un programa P tal que si el usuario suministra datos que hacen verdadera la precondición (cláusulas requiere), entonces P termina en una cantidad finita de pasos retornando un valor que hace verdadera la postcondición (cláusulas asegura).*
- 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?

Interpretando una especificación

- problema *raizCuadrada*$(x : \mathbb{R}) : \mathbb{R}$ $\{$

Interpretando una especificación

- problema *raizCuadrada*($x : \mathbb{R}$) : $\mathbb{R}$ $\{$
  requiere: $\{$$x$ debe ser mayor o igual que 0$\}$

Interpretando una especificación

- problema *raizCuadrada*($x : \mathbb{R}$) : $\mathbb{R}$ $\{$
  requiere: $\{x$ debe ser mayor o igual que $0\}$
  asegura: $\{res$ debe ser mayor o igual que $0\}$

Interpretando una especificación

- 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 $x\}$
  $\}$

Interpretando una especificación

- 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 $x\}$
  }
- ¿Qué significa esta especificación?

Interpretando una especificación

- 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 $x\}$

  }
- ¿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 debe **terminar** y el estado final cumplir $res \times res = x$ y $res \geq 0$.

Otro ejemplo

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

Otro ejemplo

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

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

Otro ejemplo

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

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

Qué sucede si ejecutamos con ...

- dividendo $= 1$ y divisor $= 0$?

Otro ejemplo

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

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

Qué sucede si ejecutamos con ...

- dividendo $= 1$ y divisor $= 0$?
- dividendo $= -4$ y divisor $= -2$, y obtenemos *res* $= 2$?

Otro ejemplo

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

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

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 $res = 0$?

Otro ejemplo

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

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

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?

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.

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

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

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

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)

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

Tipos de datos de nuestro lenguaje de especificación

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

Tipo $\mathbb{Z}$ (números enteros)

- Su conjunto base son los números enteros.

Tipo $\mathbb{Z}$ (números enteros)

- Su conjunto base son los números enteros.
- Constantes: $0 \quad ; \quad 1 \quad ; \quad -1 \quad ; \quad 2 \quad ; \quad -2 \quad ; \quad \ldots$

Tipo $\mathbb{Z}$ (números enteros)

- Su conjunto base son los números enteros.
- Constantes: $0 \quad ; \quad 1 \quad ; \quad -1 \quad ; \quad 2 \quad ; \quad -2 \quad ; \quad \ldots$
- Operaciones aritméticas:
  - $a + b$ (suma); $a - b$ (resta); abs(a) (valor absoluto)
  - $a \times b$ (multiplicación); a div b (división entera);
  - a mod b (resto de dividir a a por b), a^b o pot(a,b) (potencia)
  - a / b (división, da un valor de $\mathbb{R}$)

Tipo $\mathbb{Z}$ (números enteros)

- Su conjunto base son los números enteros.
- Constantes: $0 \quad ; \quad 1 \quad ; \quad -1 \quad ; \quad 2 \quad ; \quad -2 \quad ; \quad \ldots$
- Operaciones aritméticas:
  - $a + b$ (suma); $a - b$ (resta); abs(a) (valor absoluto)
  - $a \times b$ (multiplicación); a div b (división entera);
  - a mod b (resto de dividir a a por b), a^b o 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 <= b (menor o igual)
  - $a > b$ (mayor)
  - $a \geq b$ o a >= b (mayor o igual)
  - $a = b$ (iguales)
  - $a \neq b$ (distintos)

Tipo $\mathbb{R}$ (números reales)

- Su conjunto base son los números reales.

Tipo $\mathbb{R}$ (números reales)

- Su conjunto base son los números reales.
- Constantes: $0 \quad ; \quad 1 \quad ; \quad -7 \quad ; \quad 81 \quad ; \quad 7{,}4552 \quad ; \quad \pi \ldots$

Tipo $\mathbb{R}$ (números reales)

- Su conjunto base son los números reales.
- Constantes: $0 \quad ; \quad 1 \quad ; \quad -7 \quad ; \quad 81 \quad ; \quad 7{,}4552 \quad ; \quad \pi \ldots$
- Operaciones aritméticas:
  - Suma, resta y producto (pero no div y mod)
  - a/b (división)
  - $\log_b(a)$ (logaritmo)
  - Funciones trigonométricas

Tipo $\mathbb{R}$ (números reales)

- Su conjunto base son los números reales.
- Constantes: $0 \quad ; \quad 1 \quad ; \quad -7 \quad ; \quad 81 \quad ; \quad 7{,}4552 \quad ; \quad \pi \ldots$
- 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 `<=` b (menor o igual)
  - $a > b$ (mayor)
  - $a \geq b$ o a `>=` b (mayor o igual)
  - $a = b$ (iguales)
  - $a \neq b$ (distintos)

Tipo Bool (valor de verdad)

- Su conjunto base es $\mathbb{B} = \{\textbf{true}, \textbf{false}\}$.
- Conectivos lógicos: !, &&, $||$, con la semántica bi-valuada estándar.

Tipo Bool (valor de verdad)

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

Tipo Char (caracteres)

- Sus elementos son las letras, dígitos y símbolos.

Tipo Char (caracteres)

- Sus elementos son las letras, dígitos y símbolos.
- Constantes: ‘a‘, ‘b‘, ‘c‘, $\ldots$, ‘z‘, $\ldots$, ‘A‘, ‘B‘, ‘C‘, $\ldots$, ‘Z‘, $\ldots$, ‘0‘, ‘1‘, ‘2‘, $\ldots$, ‘9‘ (en el orden dado por el estándar ASCII).

Tipo Char (caracteres)

- Sus elementos son las letras, dígitos y símbolos.
- Constantes: $`a`, `b`, `c`, \ldots, `z`, \ldots, `A`, `B`, `C`, \ldots, `Z`, \ldots,$ $`0`, `1`, `2`, \ldots, `9`$ (en el orden dado por el estándar ASCII).
- Función ord, que numera los caracteres, con las siguientes propiedades:
  - $\mathrm{ord}(`a`) + 1 = \mathrm{ord}(`b`)$
  - $\mathrm{ord}(`A`) + 1 = \mathrm{ord}(`B`)$
  - $\mathrm{ord}(`1`) + 1 = \mathrm{ord}(`2`)$

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 $\text{char}(\text{ord}(c)) = c$.

Tipo Char (caracteres)

- Sus elementos son las letras, dígitos y símbolos.
- Constantes: $`a`, `b`, `c`, \ldots, `z`, \ldots, `A`, `B`, `C`, \ldots, `Z`, \ldots,$ $`0`, `1`, `2`, \ldots, `9`$ (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 $\text{ord}(a) < \text{ord}(b)$.

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.
- $T_0 \times T_1 \times \cdots \times T_k$: Tipo de las k-uplas de elementos de tipos T_0, T_1, ... T_k, respectivamente, donde k es fijo.

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.
- $T_0 \times T_1 \times \cdots \times T_k$: Tipo de las k-uplas de elementos de tipos T_0, T_1, ... T_k, respectivamente, donde k es fijo.
- Ejemplos:
  - $\mathbb{Z} \times \mathbb{Z}$ son los pares ordenados de enteros.

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.
- $T_0 \times T_1 \times \cdots \times T_k$: Tipo de las k-uplas de elementos de tipos T_0, T_1, ... T_k, respectivamente, donde k es fijo.
- Ejemplos:
  - $\mathbb{Z} \times \mathbb{Z}$ son los pares ordenados de enteros.
  - $\mathbb{Z} \times \mathsf{Char} \times \mathsf{Bool}$ son las triplas ordenadas con un entero, luego un carácter y luego un valor booleano.

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.
- $T_0 \times T_1 \times \cdots \times T_k$: Tipo de las k-uplas de elementos de tipos T_0, T_1, ... T_k, respectivamente, donde k es fijo.
- Ejemplos:
  - $\mathbb{Z} \times \mathbb{Z}$ son los pares ordenados de enteros.
  - $\mathbb{Z} \times \mathsf{Char} \times \mathsf{Bool}$ son las triplas ordenadas con un entero, luego un carácter y luego un valor booleano.
- *n*ésimo: $(a_0, \ldots, a_k)_m$ es el valor a_m en caso de que $0 \leq m \leq k$. Si no, está indefinido.
- Ejemplos:
  - $(7, 5)_0 = 7$
  - $(\text{‘a’}, \textbf{true}, 78)_2 = 78$

Secuencias

- **Secuencia:** Varios elementos del mismo tipo T, posiblemente repetidos, ubicados en un cierto orden.

Secuencias

- **Secuencia:** Varios elementos del mismo tipo T, posiblemente repetidos, ubicados en un cierto orden.
- $seq\langle T\rangle$ es el tipo de las secuencias cuyos elementos son de tipo T.

Secuencias

- **Secuencia:** Varios elementos del mismo tipo T, posiblemente repetidos, ubicados en un cierto orden.
- $seq\langle T \rangle$ 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;

Secuencias

- **Secuencia:** Varios elementos del mismo tipo T, posiblemente repetidos, ubicados en un cierto orden.
- $seq\langle T\rangle$ 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.

Secuencias. Notación

- Una forma de escribir un elemento de tipo $seq\langle T \rangle$ 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 $\mathbb{Z}$.

Secuencias. Notación

- Una forma de escribir un elemento de tipo $seq\langle T\rangle$ 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 $\mathbb{Z}$.
  - $\langle 1,\ 1+1,\ 3,\ 2\times 2,\ 5 \text{ mod } 2,\ 0\rangle$ es otra secuencia de $\mathbb{Z}$ (igual a la anterior).

Secuencias. Notación

- Una forma de escribir un elemento de tipo $seq\langle T\rangle$ 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.

Secuencias. Notación

- Una forma de escribir un elemento de tipo $seq\langle T\rangle$ 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.

Secuencias. Notación

- Una forma de escribir un elemento de tipo $seq\langle T\rangle$ 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 $seq\langle\mathbb{Z}\rangle$ es un tipo, podemos armar secuencias de $seq\langle\mathbb{Z}\rangle$ (secuencias de secuencias de $\mathbb{Z}$, o sea $seq\langle seq\langle\mathbb{Z}\rangle\rangle$).

Secuencias. Notación

- Una forma de escribir un elemento de tipo $\mathit{seq}\langle T\rangle$ es escribir términos de tipo T separados por comas, entre $\langle \ldots \rangle$.
  - $\langle 1, 2, 3, 4, 1, 0\rangle$ 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 $\mathit{seq}\langle\mathbb{Z}\rangle$ es un tipo, podemos armar secuencias de $\mathit{seq}\langle\mathbb{Z}\rangle$ (secuencias de secuencias de $\mathbb{Z}$, o sea $\mathit{seq}\langle \mathit{seq}\langle\mathbb{Z}\rangle\rangle$).
  - $\langle\langle 12, 13\rangle, \langle -3, 9, 0\rangle, \langle 5\rangle, \langle\rangle, \langle\rangle, \langle 3\rangle\rangle$ es un elemento de tipo $\mathit{seq}\langle \mathit{seq}\langle\mathbb{Z}\rangle\rangle$.

Secuencias bien formadas

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

- $\langle 1, 2, 3, 4, 5\rangle$?

Secuencias bien formadas

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

- $\langle 1, 2, 3, 4, 5\rangle$? Bien Formada. Tipa como $seq\langle\mathbb{Z}\rangle$ y $seq\langle\mathbb{R}\rangle$
- $\langle 1, true, 3, 4, 5\rangle$?

Secuencias bien formadas

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

- $\langle 1, 2, 3, 4, 5\rangle$? Bien Formada. Tipa como $seq\langle\mathbb{Z}\rangle$ y $seq\langle\mathbb{R}\rangle$
- $\langle 1, true, 3, 4, 5\rangle$? No está bien formada porque no es homogénea (Bool y $\mathbb{Z}$)
- $\langle 'a', 2, 3, 4, 5\rangle$?

Secuencias bien formadas

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

- $\langle 1,2,3,4,5\rangle$? Bien Formada. Tipa como $seq\langle\mathbb{Z}\rangle$ y $seq\langle\mathbb{R}\rangle$
- $\langle 1, true, 3, 4, 5\rangle$? No está bien formada porque no es homogénea (Bool y $\mathbb{Z}$)
- $\langle 'a', 2, 3, 4, 5\rangle$? No está bien formada porque no es homogénea ($Char$ y $\mathbb{Z}$)
- $\langle 'H', 'o', 'l', 'a'\rangle$?

Secuencias bien formadas

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

- $\langle 1, 2, 3, 4, 5\rangle$? Bien Formada. Tipa como $seq\langle\mathbb{Z}\rangle$ y $seq\langle\mathbb{R}\rangle$
- $\langle 1, true, 3, 4, 5\rangle$? No está bien formada porque no es homogénea (Bool y $\mathbb{Z}$)
- $\langle 'a', 2, 3, 4, 5\rangle$? No está bien formada porque no es homogénea ($Char$ y $\mathbb{Z}$)
- $\langle 'H', 'o', 'l', 'a'\rangle$? Bien Formada. Tipa como $seq\langle Char\rangle$
- $\langle true, false, true, true\rangle$?

Secuencias bien formadas

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

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

Secuencias bien formadas

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

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

Secuencias bien formadas

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

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

Secuencias bien formadas

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

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

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:
  - $|\langle\rangle| =$

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:
  - $|\langle\rangle| = 0$
  - $|\langle 'H', 'o', 'l', 'a' \rangle| =$

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:
  - $|\langle\rangle| = 0$
  - $|\langle 'H', 'o', 'l', 'a' \rangle| = 4$
  - $|\langle 1, 1, 2\rangle| =$

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:
  - $|\langle\rangle| = 0$
  - $|\langle 'H', 'o', 'l', 'a' \rangle| = 4$
  - $|\langle 1, 1, 2\rangle| = 3$

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.

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:
  - $\langle 'H', 'o', 'l', 'a' \rangle[0] =$

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:
  - $\langle 'H','o','l','a'\rangle[0] = 'H'$
  - $\langle 'H','o','l','a'\rangle[1] =$

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:
  - $\langle 'H',' o',' l',' a'\rangle[0] = 'H'$
  - $\langle 'H',' o',' l',' a'\rangle[1] = 'o'$
  - $\langle 'H',' o',' l',' a'\rangle[2] =$

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:
  - $\langle 'H',' o',' l',' a'\rangle[0] = 'H'$
  - $\langle 'H',' o',' l',' a'\rangle[1] = 'o'$
  - $\langle 'H',' o',' l',' a'\rangle[2] = 'l'$
  - $\langle 'H',' o',' l',' a'\rangle[3] =$

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:
  - $\langle 'H',' o',' l',' a'\rangle[0] = 'H'$
  - $\langle 'H',' o',' l',' a'\rangle[1] = 'o'$
  - $\langle 'H',' o',' l',' a'\rangle[2] = 'l'$
  - $\langle 'H',' o',' l',' a'\rangle[3] = 'a'$
  - $\langle 1, 1, 1, 1\rangle[0] =$

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:
  - $\langle 'H',' o',' l',' a'\rangle[0] = 'H'$
  - $\langle 'H',' o',' l',' a'\rangle[1] = 'o'$
  - $\langle 'H',' o',' l',' a'\rangle[2] = 'l'$
  - $\langle 'H',' o',' l',' a'\rangle[3] = 'a'$
  - $\langle 1, 1, 1, 1\rangle[0] = 1$
  - $\langle\rangle[0] =$

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)
  - $\langle 1, 1, 1, 1\rangle[7] =$

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)

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:
  - $(1,' b') \in \langle(1,' a'),(2,' b'),(3,' c'),(1,' b')\rangle$?

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**
  - $(1,' b') \in \langle(1,' a'), (2,' b'), (3,' c'), (3,' b')\rangle$?

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**

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:

- $\langle 1, 2, 3, 4 \rangle = \langle 1, 2, 3, 4 \rangle$?

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
- $\langle \rangle = \langle \rangle$?

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
- $\langle 4, 4, 4 \rangle = \langle 4, 4, 4 \rangle$?

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
- $\langle 1, 2, 3, 4, 5 \rangle = \langle 1, 2, 3, 4 \rangle$?

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
- $\langle 1, 2, 3, 4, 5 \rangle = \langle 1, 2, 4, 5, 6 \rangle$?

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
- $\langle 1, 2, 3, 5, 4 \rangle = \langle 1, 2, 3, 4, 5 \rangle$?

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

Funciones con secuencias

Cabeza o Head

- Cabeza: $head(a : seq\langle T\rangle) : T$
  - 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:
  - $head(\langle' H',' o',' l',' a'\rangle) =$

Funciones con secuencias

Cabeza o Head

- Cabeza: $head(a : seq\langle T\rangle) : T$
  - 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:
  - $head(\langle 'H', 'o', 'l', 'a' \rangle) = 'H'$
  - $head(\langle 1, 1, 1, 1\rangle) =$

Funciones con secuencias

Cabeza o Head

- Cabeza: $head(a : seq\langle T\rangle) : T$
  - 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:
  - $head(\langle 'H', 'o', 'l', 'a'\rangle) = 'H'$
  - $head(\langle 1, 1, 1, 1\rangle) = 1$
  - $head(\langle\rangle) =$

Funciones con secuencias

Cabeza o Head

- Cabeza: $head(a : seq\langle T\rangle) : T$
  - 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:
  - $head(\langle 'H', 'o', 'l', 'a' \rangle) = 'H'$
  - $head(\langle 1, 1, 1, 1\rangle) = 1$
  - $head(\langle\rangle) = \perp$ (Indefinido)

Funciones con secuencias

Cola o Tail

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

Funciones con secuencias

Cola o Tail

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

Funciones con secuencias

Cola o Tail

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

Funciones con secuencias

Cola o Tail

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

Funciones con secuencias

Cola o Tail

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

Funciones con secuencias

Agregar al principio o addFirst

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

Funciones con secuencias

Agregar al principio o addFirst

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

Funciones con secuencias

Agregar al principio o addFirst

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

Funciones con secuencias

Agregar al principio o addFirst

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

Funciones con secuencias

Concatenación o concat

- Concatenación: $concat(a : seq\langle T\rangle, b : seq\langle T\rangle) : seq\langle T\rangle$
  - Es una secuencia con los elementos de a, seguidos de los de b.
  - Notación: $concat(a, b)$ se puede escribir $a +\!\!+\, b$.

Funciones con secuencias

Concatenación o concat

- Concatenación: $concat(a : seq\langle T\rangle, b : seq\langle T\rangle) : seq\langle T\rangle$
  - Es una secuencia con los elementos de a, seguidos de los de b.
  - Notación: $concat(a, b)$ se puede escribir $a +\!\!+\, b$.
- Ejemplos:
  - $concat(\langle 'H', 'o'\rangle, \langle 'l', 'a'\rangle) =$

Funciones con secuencias

Concatenación o concat

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

Funciones con secuencias

Concatenación o concat

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

Funciones con secuencias

Concatenación o concat

- Concatenación: $concat(a : seq\langle T\rangle, b : seq\langle T\rangle) : seq\langle T\rangle$
  - Es una secuencia con los elementos de a, seguidos de los de b.
  - Notación: $concat(a, b)$ se puede escribir $a ++ b$.

- Ejemplos:
  - $concat(\langle 'H', 'o'\rangle, \langle 'l', 'a'\rangle) = \langle 'H', 'o', 'l', 'a'\rangle$
  - $concat(\langle 1, 2\rangle, \langle 3, 4\rangle) = \langle 1, 2, 3, 4\rangle$
  - $concat(\langle\rangle, \langle\rangle) = \langle\rangle$
  - $concat(\langle 2, 3\rangle, \langle\rangle) =$

Funciones con secuencias

Concatenación o concat

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

Funciones con secuencias

Concatenación o concat

- Concatenación: $concat(a : seq\langle T\rangle, b : seq\langle T\rangle) : seq\langle T\rangle$
  - Es una secuencia con los elementos de a, seguidos de los de b.
  - Notación: $concat(a, b)$ se puede escribir $a +\!+\, b$.

- Ejemplos:
  - $concat(\langle 'H', 'o'\rangle, \langle 'l', 'a'\rangle) = \langle 'H', 'o', 'l', 'a'\rangle$
  - $concat(\langle 1, 2\rangle, \langle 3, 4\rangle) = \langle 1, 2, 3, 4\rangle$
  - $concat(\langle\rangle, \langle\rangle) = \langle\rangle$
  - $concat(\langle 2, 3\rangle, \langle\rangle) = \langle 2, 3\rangle$
  - $concat(\langle\rangle, \langle 5, 7\rangle) = \langle 5, 7\rangle$

Funciones con secuencias

Subsecuencia o subseq

- Subsecuencia: $subseq(a : seq\langle T\rangle, d, h : \mathbb{Z}) : seq\langle T\rangle$
  - 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:
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 1) =$

Funciones con secuencias

Subsecuencia o subseq

- Subsecuencia: $subseq(a : seq\langle T\rangle, d, h : \mathbb{Z}) : seq\langle T\rangle$
  - 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:
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 1) = \langle 'H'\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 4) =$

Funciones con secuencias

Subsecuencia o subseq

- Subsecuencia: $subseq(a : seq\langle T\rangle, d, h : \mathbb{Z}) : seq\langle T\rangle$
  - 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:
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 1) = \langle 'H'\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 4) = \langle 'H', 'o', 'l', 'a'\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 2, 2) =$

Funciones con secuencias

Subsecuencia o subseq

- Subsecuencia: $subseq(a : seq\langle T\rangle, d, h : \mathbb{Z}) : seq\langle T\rangle$
  - 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:
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 1) = \langle 'H'\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 4) = \langle 'H', 'o', 'l', 'a'\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 2, 2) = \langle\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, -1, 3) =$

Funciones con secuencias

Subsecuencia o subseq

- Subsecuencia: $subseq(a : seq\langle T\rangle, d, h : \mathbb{Z}) : seq\langle T\rangle$
  - 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:
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 1) = \langle 'H'\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 4) = \langle 'H', 'o', 'l', 'a'\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 2, 2) = \langle\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, -1, 3) = \perp$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 10) =$

Funciones con secuencias

Subsecuencia o subseq

- Subsecuencia: $subseq(a : seq\langle T\rangle, d, h : \mathbb{Z}) : seq\langle T\rangle$
  - 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:
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 1) = \langle 'H'\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 4) = \langle 'H', 'o', 'l', 'a'\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 2, 2) = \langle\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, -1, 3) = \perp$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 10) = \perp$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 3, 1) =$

Funciones con secuencias

Subsecuencia o subseq

- Subsecuencia: $subseq(a : seq\langle T\rangle, d, h : \mathbb{Z}) : seq\langle T\rangle$
  - 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:
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 1) = \langle 'H'\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 4) = \langle 'H', 'o', 'l', 'a'\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 2, 2) = \langle\rangle$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, -1, 3) = \perp$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 0, 10) = \perp$
  - $subseq(\langle 'H', 'o', 'l', 'a'\rangle, 3, 1) = \perp$

Funciones con secuencias

Modificar un valor o setAt

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

Funciones con secuencias

Modificar un valor o setAt

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

Funciones con secuencias

Modificar un valor o setAt

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

Funciones con secuencias

Modificar un valor o setAt

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

Operaciones sobre secuencias

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

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.

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.

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.

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 $\mathit{Renombre}\ T_1 = T_2$

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 $\mathit{Renombre}\ T_1 = T_2$
- 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: $\mathit{Renombre}\ \mathit{Usuario} = \mathbb{Z} \times \mathit{seq}\langle \mathit{Char} \rangle$

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 $\mathit{Renombre}\ T_1 = T_2$
- 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: $\mathit{Renombre}\ \mathit{Usuario} = \mathbb{Z} \times \mathit{seq}\langle \mathit{Char} \rangle$
  - 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: $\mathit{Renombre}\ \mathit{Publicación} = \mathit{Usuario} \times \mathit{seq}\langle \mathit{Char} \rangle \times \mathit{seq}\langle \mathit{Usuario} \rangle$

Problemas comunes de las especificaciones

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

Sobre-especificación

Sobre-especificación

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

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.

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: $\{res \neq x\}$
  }

Sub-especificación

- Consiste en dar una precondición más restrictiva de lo realmente necesario, o bien una postcondición más débil de la que se necesita.

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).

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

  … en vez de:

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

Modularización

Partiendo un problema en problemas mas chicos

Dadas dos secuencias, queremos saber si uno es una una permutación[1] de la otra secuencia:
¿Cuándo será una secuencia permutación de la otra?

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

Modularización

Partiendo un problema en problemas mas chicos

Dadas dos secuencias, queremos saber si uno es una una permutación[1] 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

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

Modularización

Partiendo un problema en problemas mas chicos

Dadas dos secuencias, queremos saber si uno es una una permutación[1] 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 $esPermutacion(s1, s2 : seq\langle T\rangle)$: Bool {
  asegura: $\{res = true \leftrightarrow$ para cada elemento es cierto que tiene la misma cantidad de apariciones en $s1$ y $s2$ $\}$
}

Pero... falta algo...

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

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?

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

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*$(s : seq\langle T\rangle, e : T) : \mathbb{Z}$ {
  asegura $\{res =$ la cantidad de veces que el elemento e aparece en la lista s $\}$
}

Recapitulando

Partiendo un problema en problemas mas chicos

Dadas dos secuencias, queremos saber si uno es una una permutación[1] de la otra secuencia:

problema $esPermutacion(s1, s2 : seq\langle T\rangle)$: Bool {
asegura: $\{res = true \leftrightarrow$ (para cada elemento e de T, se cumple que $(cantidadDeApariciones(s1, e) = cantidadDeApariciones(s2, e)))\}$
}

Donde...

problema $cantidadDeApariciones(s : seq\langle T\rangle, e : T) : \mathbb{Z}$ {
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!

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

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

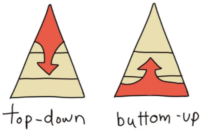
Esto favocere muchos aspectos de calidad como:

- 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)

Modularización

Top Down versus Bottom Up

También es aplicable a la especificación de problemas:



problema $esPermutacion(s1, s2 : seq\langle T\rangle)$: Bool {
asegura: $\{res = true \leftrightarrow$(para cada elemento e de T, se cumple que $(cantidadDeApariciones(s1, e) = cantidadDeApariciones(s2, e)))\}$
}

problema $cantidadDeApariciones(s : seq\langle T\rangle, e : T) : \mathbb{Z}$ {
asegura $\{res =$ la cantidad de veces que el elemento e aparece en la lista s $\}$
}

¿Lo encaramos Top Down o Bottom Up?