Resumen Teórico de IP/Algo I

Sección 0: Conceptos básicos

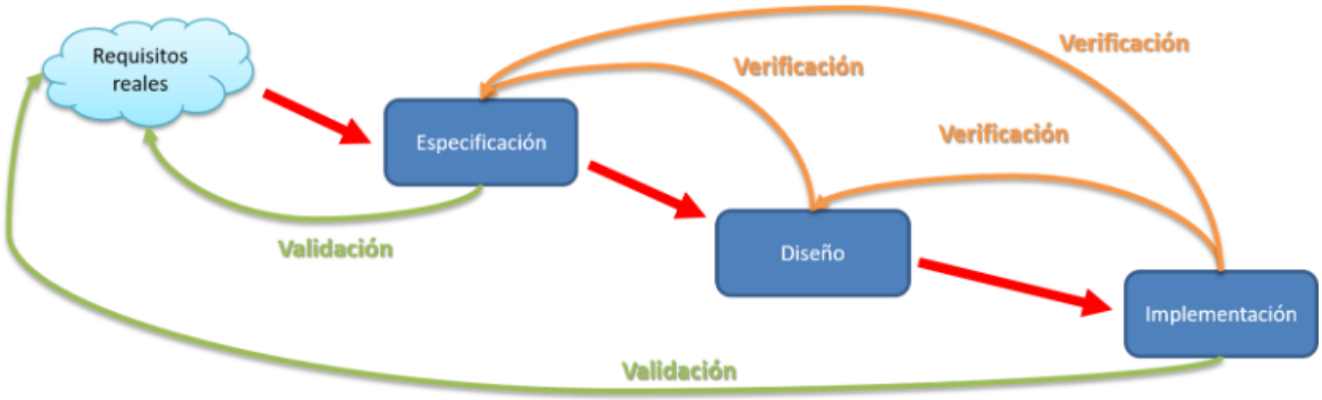
¿Qué es una computadora, un programa y un algoritmo?

Una computadora es una máquina que procesa información automáticamente de acuerdo con un programa almacenado. Un programa es la descripción de un algoritmo en un lenguaje de programación. Un algoritmo es la descripción de los pasos precisos para resolver un problema a partir de datos de entrada adecuados.

Especificación, algoritmo y programa

Una especificación es una descripción (a menudo escrita en lenguaje formal) del **problema** a resolver. Se trata de un contrato entre dos partes (cliente/programador), en el que se establecen propiedades de los datos de entrada (requiere) y de la solución (asegura). Nos habla del **qué** debemos solucionar.

El algoritmo es la descripción de la **solución** escrita para **humanos**. El programa es la descripción de la solución para ser ejecutada en una **computadora**.



Modularización y encapsulamiento

La modularización implica dividir un sistema en módulos autónomos, facilitando el desarrollo y la reutilización del código. El encapsulamiento oculta los detalles internos de un objeto y expone solo una interfaz pública para interactuar con él, mejorando la seguridad y el mantenimiento del código.

Sección 1: Lógica y Especificación

Partes de una especificación

- Encabezado

- Precondiciones (requiere): Condiciones sobre los parámetros o datos de entrada que deben cumplirse para que la implementación cumpla los asegura.
- Modifica (introducido en programación imperativa con la transformación de estados)
- Postcondiciones (asegura): Condiciones sobre los parámetros de salida y entrada/salida.

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

Sub-especificación y sobre-especificación

La 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).

La 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.

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. Todos los tipos tienen un elemento distinguido: ⊥ o Indef.

Tipos de datos en el lenguaje de especificación de la materia

- Básicos:
  - Enteros (Z)
  - Reales (R)
  - Booleanos (Bool)
  - Carácteres (Char)
- Enumerados. `enum Nombre {CONSTANTES}`
- Uplas. `T0 X T1 X...X Tk`
- Secuencias. `seq<T>`

Predicados

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

`pred p(argumentos){fórmula}`

Expresiones condicionales

Función que elige entre dos elementos del mismo tipo, según una fórmula lógica (guarda):

- Si la guarda es verdadera, elige el primero
- Si no, elige el segundo

`IfThenElseFi ⟨Z⟩ (a > b, a, b)`

`if a > b then a else b fi`

Lógica proposicional

En la lógica proposicional, las proposiciones (fórmulas) son consideradas como unidades indivisibles que pueden ser verdaderas o falsas, y se utilizan reglas y símbolos para analizar y manipular su estructura lógica.

- Símbolos: True , False , ¬ , ∧ , ∨ , → , ↔ , (,)
- Variables proposicionales: p, q, r, s, ... (hay infinitas)
- Reglas para que una fórmula sea *bien formada*:
  1. True y False son fórmulas.
  2. Cualquier variable proposicional es una fórmula.
  3. Si A es una fórmula, ¬A es una fórmula (La negación de una fórmula es una fórmula).
  4. Si A1, A2, ... , An son fórmulas, (A1 ∧ A2 ∧ ⋯ ∧ An) es una fórmula (La conjunción de fórmulas es una fórmula)
  5. Si A1, A2, ... , An son fórmulas, (A1 ∨ A2 ∨ ⋯ ∨ An) es una fórmula (La disyunción de fórmulas es una fórmula)
  6. Si A y B son fórmulas, (A → B) es una fórmula (La implicancia entre fórmulas es una fórmula)
  7. Si A y B son fórmulas, (A ↔ B) es una fórmula (La igualdad de valores de verdad (True o False) entre 2 fórmulas es una fórmula)

Semántica clásica

- Dos valores de verdad: verdadero (V) y falso (F).
- Conociendo el valor de verdad de las variables proposicionales de una fórmula podemos saber el valor de verdad de la fórmula (se usan tablas de verdad para ello).

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

Semántica trivaluada o secuencial

- 3 valores de verdad: verdadero (V), falso (F) e indefinido (⊥).
- Se evalúa de izquierda a derecha y si se puede deducir el valor de verdad (aunque el resto esté indefinido) se termina la evaluación.
- Nuevos operadores lógicos:
  - Y-luego/conditional and/cand: ∧L
  - O-luego/conditional or/cor: ∨L
  - Implicación fuerte/trivaluada: →L

Tautologías, contradicciones y contingencias

Una fórmula es:

- **Tautología** si siempre toma el valor de V para todas la posibles combinaciones de valores de verdad de sus variables proposicionales.
- **Contradicción** si siempre toma el valor de F para todas las posibles combinaciones de valores de verdad de sus variables proposicionales.
- **Contingencia **si no es ni tautología ni contradicción.

Equivalencia entre fórmulas (teoremas)

1. Doble negación.
   - ¬¬p ↔ p
2. Idempotencia.
   - (p ∧ p) ↔ p
   - (p ∨ p) ↔ p
3. Asociatividad.

- ((p ∧ q) ∧ r) ↔ (p ∧ (q ∧ r))
- ((p ∨ q) ∨ r) ↔ (p ∨ (q ∨ r))

4. Conmutatividad.
   - (p ∧ q) ↔ (q ∧ p)
   - (p ∨ q) ↔ (q ∨ p)
5. Distributividad.
   - (p ∧ (q ∨ r)) ↔ ((p ∧ q) ∨ (p ∧ r))
   - (p ∨ (q ∧ r)) ↔ ((p ∨ q) ∧ (p ∨ r))
6. Reglas de Morgan.
   - ¬(p ∧ q) ↔ (¬p ∨ ¬q)
   - ¬(p ∨ q) ↔ (¬p ∧ ¬q)
7. Ley del condicional.
   - (p -> q) <=> (¬p ∨ q)

Relación de fuerza

Decimos que **A es más fuerte que B** o que **A fuerza a B** o que **B es más débil que A** cuando **(A → B) es tautología**.

False es la fórmula más fuerte de todas y True es la más débil.

Lógica de primer orden (LPO) *(se trata de una abstracción)*

Cuantificadores, variable libre y variable ligada

Los cuantificadores permiten predicar sobre algunos o todos los elementos de un conjunto:

- Para todo (cuantificador universal): (∀x : T) P(x) *Afirma que todos los elementos de tipo T cumplen la propiedad P.*
- Existe (cuantificador existencial): (∃x : T) P(x) *Afirma que al menos un elemento de tipo T cumple la propiedad P.*

Una variable es **libre** cuando no está ligada a ningún cuantificador (en los ejemplos anteriores x está **ligada**).

La negación de un cuantificador universal es un cuantificador existencial, y viceversa. Además, el cuantificador universal generaliza la **conjunción** *(TODOS LOS ELEMENTOS DEBEN CUMPLIR ALGO)* y el existencial la **disyunción** *(AL MENOS UN ELEMENTO DEBE CUMPLIR ALGO).*

Sección 2: Paradigma funcional y paradigma imperativo

Un paradigma de programación es un enfoque o estilo general para diseñar, estructurar y resolver problemas de programación. Existen diversos paradigmas de programación. Comúnmente se los divide en dos grandes grupos:

- Programación declarativa: Son lenguajes donde el programador le indicará a la máquina **lo que quiere hacer y el resultado que desea**, pero no necesariamente el cómo hacerlo. Describe un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución.
- Programación imperativa: Son lenguajes en los que el programador debe precisarle a la máquina de forma exacta el **proceso que quiere realizar. **Describe la programación como una secuencia de instrucciones o comandos que cambian el estado de un programa.

Paradigma funcional

Paradigma imperativo

Sección 3: Testing