Algoritmos y estructuras de datos II
Especificación

Carlos Gustavo Lopez Pombo
(Charlie)

Departamento de Computación,
Facultad de ciencias exactas y naturales,
Universidad de Buenos Aires

Resolución de problemas

En esta materia vamos a estudiar **algoritmos** y estructuras de datos

Algoritmo: secuencia ordenada y finita de pasos que realiza un cómputo. Dado un **estado inicial**, una vez ejecutados todos los pasos, se llega a un **estado final**

Resolución de problemas

En el contexto de la materia, un **problema** es una descripción abstracta de un objetivo a alcanzar a partir de ciertas condiciones iniciales.

Y su **solución** será un algoritmo que satisfaga esa descripción, es decir, que dados los datos en cuestión, satisfaciendo las condiciones iniciales, alcance el objetivo descripto.

Resolución de problemas

Se acuerdan de Algo 1

De esta forma, la especificación de un problema puede ser entendida como un **estado inicial** (condiciones que deben satisfacerse sobre los datos) y un **estado final** (condiciones que deben satisfacerse sobre el resultado)

Ejemplo: suma de naturales

Descripción: dados a y b números naturales se pretende saber cuál es su suma

Estado inicial: {true}

Estado final: {suma (a, b) = a + b}

Algoritmo: return a + b;

Ejemplo: división de naturales

Descripción: dados a y b números naturales se pretende saber cuál es su división entera

Estado inicial: {b > 0}

Estado final: {div (a, b) = a / b}

Algoritmo: return a / b;

Ejemplo: bla, bla, bla

Descripción: Dados cuatro reales encontrar al entero más chico que esté por encima del mínimo cuadrado perfecto mayor que el mínimo de los dos primeros pero menor que la suma de los otros dos

Estado inicial: {a>0 && b>0}

Estado final: {blablabla (a,b,c,d) =

$$\min\{w \mid w > \min\{x \mid \text{sqrt}(x)/|\text{sqrt}(x)| = 1 \;\&\&\; \min(a, b) < x < c+d\}\}$$

Algoritmo: Ya no parece tan obvio cómo escribir un programa que resuelva esto, ¿no?

Ejemplo: restaurant

Descripción: El dueño de un restaurant quiere asegurarse de que los pedidos de sus clientes sean atendidos con prolijidad. Los mozos llevan los pedidos hasta la cocina donde los colocan en un rotador de tarjetas. Cuando el cocinero se libera, saca la primera tarjeta y prepara el plato allí indicado. El dueño quiere saber cuál es el próximo plato a preparar, cuántos pedidos atiende el cocinero cada día, y cuál fue el día con menos pedidos

Estado inicial: ¿?

Estado final: ¿?

Algoritmo: ¿?

Conclusión

¡Especificar es una estafa!

Si solo puedo ser riguroso especificando cuando los problemas son de juguete, entonces esto no tiene ninguna utilidad práctica

¡NO!

Las **herramientas** que utilizamos deben mantener alguna relación con los **problemas** que queremos resolver...

Tipos Abstractos de Datos

Un TAD caracteriza en forma **rigurosa** y **formal** los aspectos "relevantes" de un fragmento de la realidad.

La palabra **tipo** no es inocente. Identifica la colección de "cosas" que comparten el comportamiento descripto.

Tipos Abstractos de Datos

¿Qué es un TAD?

Desde el punto de vista formal: es una herramienta matemática. Eso es bueno porque nos da la rigurosidad y formalidad que necesitamos para entender claramente qué es lo que hay que hacer, es decir, para describir el problema que se pretende resolver.

Tipos Abstractos de Datos

¿Qué es un TAD?

Desde el punto de vista práctico: es una herramienta poderosa y flexible que como veremos más adelante nos va a permitir describir una enorme cantidad de problemas.

Tipos Abstractos de Datos

¿Qué es un TAD?

Desde el punto de vista histórico: es uno de los primeros intentos por abordar la problemática de describir formalmente el comportamiento de un artefacto de software.

Tipos Abstractos de Datos

Volviendo a nuestro problema...

El dueño de un **restaurant** quiere asegurarse de que los pedidos de sus clientes sean atendidos con prolijidad. Los mozos llevan los pedidos hasta la cocina donde los colocan en un rotador de tarjetas. Cuando el cocinero se libera, saca la primera y prepara el **plato** allí indicado. El dueño quiere saber cuál es el próximo plato a preparar, cuántos pedidos atiende el cocinero cada **día**, cuál fue el plato más pedido y cuál fue el día con menos pedidos

Tipos Abstractos de Datos

Volviendo a nuestro problema...

Sabemos que:

Los **días** pasan puesto que se quiere saber qué se pidió en cada uno

Los **platos** son diferentes porque se quiere saber el plato más pedido y nada indica que el menú cambie a lo largo del tiempo

Los **restaurants** reciben pedidos (de entre los platos de los que dispone), atienden dichos pedidos en orden de llegada, etc.

Tipos Abstractos de Datos

Esta separación permite pensar en forma modular el problema, a partir de problemas de menor complejidad cuyas soluciones se pueden componer para resolver el problema original...

Resta ahora saber cómo caracterizar formalmente el comportamiento de los tres tipos identificados (los platos, los días y los restaurantes).

Lógica (sí, lógica otra vez)

Lógica trivaluada: se tienen valores de verdad true (verdadero), false (falso) y ⊥ (indefinido)...

f(a, b, c) = a/b == c evalúa a ⊥ si se aplica a 4, 0 y 2

Lógica (sí, lógica otra vez)

Lógica trivaluada: se tienen valores de verdad true (verdadero), false (falso) y ⊥ (indefinido)...

¡Tenemos un valor más que podemos usar!

Lógica (sí, lógica otra vez)

Lógica trivaluada: se tienen valores de verdad true (verdadero), false (falso) y ⊥ (indefinido)...

Indefinido significa que algo **NO** puede ser evaluado computacionalmente.

Lógica (sí, lógica otra vez)

Lógica trivaluada: se tienen valores de verdad true (verdadero), false (falso) y ⊥ (indefinido)...

Introducimos nuevos operadores lógicos que consideran esta nueva situación al ser evaluados.

Lógica (sí, lógica otra vez)

Operadores básicos de la lógica trivaluada

x	y	¬ x	x ∨ y	x ∧ y	x ∨L y	x ∧L y
0	0	I	0	0	0	0
0	I	I	I	0	I	0
0	⊥	I	⊥	⊥	⊥	0
I	0	0	I	0	I	0
I	I	0	I	I	I	I
I	⊥	0	⊥	⊥	I	⊥
⊥	0	⊥	⊥	⊥	⊥	⊥
⊥	I	⊥	⊥	⊥	⊥	⊥
⊥	⊥	⊥	⊥	⊥	⊥	⊥

Lógica (sí, lógica otra vez)

Lógica trivaluada: se tienen valores de verdad true (verdadero), false (falso) y ⊥ (indefinido)...

f(a, b, c) = (b != 0) ∧L (a/b == c) nunca "evalúa" a ⊥

Lógica (sí, lógica otra vez)

Logica de primer orden con ≡ (igualdad)

Sean {T_i | 0<=i<=n} un conjunto de tipos, luego todo símbolo de función f tiene un **tipo** que se denota como:

$$f: T_{i_1} x \ldots x T_{i_j} \rightarrow T_{i_{j+1}},$$

que quiere decir que f toma como argumentos j elementos de los tipos T_{i_1} a T_{i_j} respectivamente y retorna un elemento de $T_{i_{j+1}}$

Lógica (sí, lógica otra vez)

Logica de primer orden con ≡ (igualdad)

Sean X un conjunto de variables (cada una con un tipo asociado) y F = {f_i | 0<=i<=m} símbolos de función, se define Term(X, F) como el conjunto de **términos** que satisface:

- X está incluido en Term(X, F)

- dada f: $T_{i_1} x \ldots x T_{i_j} \rightarrow T_{i_{j+1}}$ que pertenece a F y $t_1, \ldots, t_j$ en Term(X, F), entonces $f(t_1, \ldots, t_j)$ pertenece a Term(X, F)

Lógica (sí, lógica otra vez)

Logica de primer orden con ≡ (igualdad)

Sea Term(X, F) un conjunto términos, se define el conjunto de fórmulas bien formadas Form(X, F) como:

- para todo t, t' en Term(X, F), t ≡ t' pertenece a Form(X, F)

- dados P, Q en Form(X, F), y x en X, entonces

¬P, P∨Q, P∧Q, P∨L Q, P∧L Q, (∃x)P y (∀x)P pertenecen a Form(X, F)

Lógica (sí, lógica otra vez)

Logica de primer orden con ≡ (igualdad)

Dado X un conjunto de símbolos de variable y F un conjunto de símbolos de función, se denomina **sentencia** a una fórmula sin variables libres.

Sea P en Form(X, F), la interpretación de las sentencias cuantificadas es:

$(\exists x{:}T)P(x)=P(t1)\vee P(t2)\vee\ldots$ (con t1,t2,... todos los términos de T)

$(\forall x{:}T)P(x)=P(t1)\wedge P(t2)\wedge\ldots$ (con t1,t2,... todos los términos de T)

Lógica (sí, lógica otra vez)

Logica de primer orden con ≡ (igualdad)

En una gran cantidad de casos se pretende evaluar sobre un subconjunto determinado de los términos de un tipo.

Sea P y Q en Form(X, F), luego

$(\exists x{:}T)(P(x) \wedge Q(x))$ vale si existe un término en T tal que se encuentra en el subconjunto caracterizado por P y además satisface la propiedad Q

Lógica (sí, lógica otra vez)

Logica de primer orden con ≡ (igualdad)

En una gran cantidad de casos se pretende evaluar un predicado sobre un subconjunto determinado de los términos de un tipo pues se indefinite para el resto.

Sea P y Q en Form(X, F), luego

(∃x:T)(P(x) ∧L Q(x)) vale si existe un término en T tal que se encuentra en el subconjunto caracterizado por P y además satisface la propiedad Q

Lógica (sí, lógica otra vez)

Logica de primer orden con ≡ (igualdad)

En una gran cantidad de casos se pretende evaluar sobre un subconjunto determinado de los términos de un tipo.

Sea P y Q en Form(X, F), luego

$(\forall x{:}T)(\neg P(x) \vee Q(x))$ vale si la totalidad de los términos en T que se encuentran en el subconjunto caracterizado por P además satisface la propiedad Q

Lógica (sí, lógica otra vez)

Logica de primer orden con ≡ (igualdad)

En una gran cantidad de casos se pretende evaluar un predicado sobre un subconjunto determinado de los términos de un tipo pues se indefinite para el resto.

Sea P y Q en Form(X, F), luego

(∀x:T)(¬P(x) ∨L Q(x)) vale si la totalidad de los términos en T que se encuentran en el subconjunto caracterizado por P además satisface la propiedad Q

Lógica (sí, lógica otra vez)

Logica de primer orden con ≡ (igualdad)

Para simplificar la notación usaremos las siguientes expansiones:

$(\exists x{:}T)(P(x) \wedge_L Q(x)) \equiv (\exists x{:}T, P(x))\ Q(x) \equiv (\exists x{:}T \mid P(x))\ Q(x)$

$(\forall x{:}T)(\neg P(x) \vee_L Q(x)) \equiv (\forall x{:}T, P(x))\ Q(x) \equiv (\forall x{:}T \mid P(x))\ Q(x)$

Lógica (sí, lógica otra vez)

Logica de primer orden con ≡ (igualdad)

- Existen naturales pares menores que 33.
- Todos las secuencias de longitud par se pueden escribir como la concatenación de otras dos secuencias.
- Todos los números primos tienen un elemento mayor y otro menor.
- Cada secuencia de naturales puede ser extendida en un elemento en tantas secuencias como naturales.
- Todo Zn tiene su neutro aditivo.
- Hay un neutro aditivo e para todo Zn.

Tipos Abstractos de Datos

Algunas ventajas de la teoría detrás de los tipos abstractos de datos:

- Tiene fundamentos matemáticos sólidos en el campo del álgebra
- Toda la teoría es definible en forma homogénea, es decir que no se requieren tipos primitivos ni construcciones ad-hoc

Tipos Abstractos de Datos

Algunas preguntas que surgen...

¿Qué comportamientos identificamos?

¿Cómo los expresamos formalmente (me refiero a hacerlo en un sentido matemático)?

¿Cómo estructuramos la información?

Tipos Abstractos de Datos

Veámoslo con un ejemplo

Definamos el tipo de los valores booleanos...

¡Usando un TAD!

Tipos Abstractos de Datos

Un poco de notación

Géneros. Los géneros (en general va a haber solo uno, pero podrían ser más) son los nombres de los tipos que un TAD define.

Tipos Abstractos de Datos

Un poco de notación

Exporta. Detalla que operaciones y géneros se dejan a disposición de los usuarios del tipo.

Tipos Abstractos de Datos

Un poco de notación

Generadores. Son operaciones que permiten construir términos del tipo. Un conjunto de generadores es completo si todo término del tipo puede ser construido como una combinación de ellos.

Tipos Abstractos de Datos

Un poco de notación

Observadores. Son aquellas operaciones que nos permiten, utilizadas en conjunto, distinguir si dos instancias del tipo son iguales o no.

Tipos Abstractos de Datos

Un poco de notación

Otras Operaciones. Son aquellas operaciones adicionales que caracterizan algún comportamiento esperado del tipo, y que tienen que ser consistentes con el comportamiento declarado por los observadores básicos.

Tipos Abstractos de Datos

Un poco de notación

Axiomas. Son las reglas que definen matemáticamente el comportamiento de las operaciones identificadas.

Tipos Abstractos de Datos

(Gateando)

TAD Bool
Generos: bool
Generadores:
    true: . -> bool
    false: . -> bool
Otras Operaciones:
    not: bool -> bool
    or: bool x bool -> bool
    and: bool x bool -> bool
...

Axiomas:
not (true) ≡ false
not (false) ≡ true
true or x ≡ true
false or x ≡ x
x and y ≡ not (not (x) or not (y))

Tipos Abstractos de Datos

Avancemos un poco

A ver cómo nos va con el tipo de los números naturales...

Tipos Abstractos de Datos

Un poco más de notación

Usa. Denota la inclusión de géneros y operaciones de otro tipo que son necesarios para resolver el problema.

Tipos Abstractos de Datos

(Aprendiendo a caminar)

TAD Nat
Usa: Bool
Generos: nat
Observadores Básicos:
0?: nat -> bool
pred: nat n -> nat

Generadores:
0: . -> nat
suc: nat -> nat

Otras Operaciones:
=: nat x nat -> bool, >: nat x nat -> bool, ...
suma: nat x nat -> nat
prod: nat x nat -> nat
...

0?(n) ≡ false

Axiomas:
0? (0) ≡ true
0? (suc(y)) ≡ false
pred (suc(y)) ≡ y
...

Tipos Abstractos de Datos

(Aprendiendo a caminar)

TAD Nat
Usa: Bool
Generos: nat
Observadores Básicos:
0?: nat -> bool
pred: nat n -> nat
Generadores:
0: . -> nat
suc: nat -> nat
Otras Operaciones:
=: nat x nat -> bool, >: nat x nat -> bool, ...
suma: nat x nat -> nat
prod: nat x nat -> nat
...

Axiomas:
0?(n) ≡ false

x = y ≡ if (0?(x) and 0?(y)) then true
else if (not 0?(x) and not 0?(y) then
pred(x) = pred(y)
else false

...
suma (x, 0) ≡ x
suma (x, suc(y)) ≡ suc(suma(x, y))
prod (x, 0) ≡ 0
prod (x, suc(y)) ≡ suma (x, prod (x, y))
...

Repaso

- Introdujimos a nivel de la intuición lo que haremos a lo largo de la primera parte de la materia
- Repasamos un poco de lógica
- Definimos la herramienta que usaremos para especificar
- Describimos formalmente dos de los tipos básicos que usaremos de aquí en más.