Astrazione nella progettazione

L'astrazione

Astrarre: dal lat. *abstrahere* = 'trascinare via', dal verbo *trahere* = trascinare.

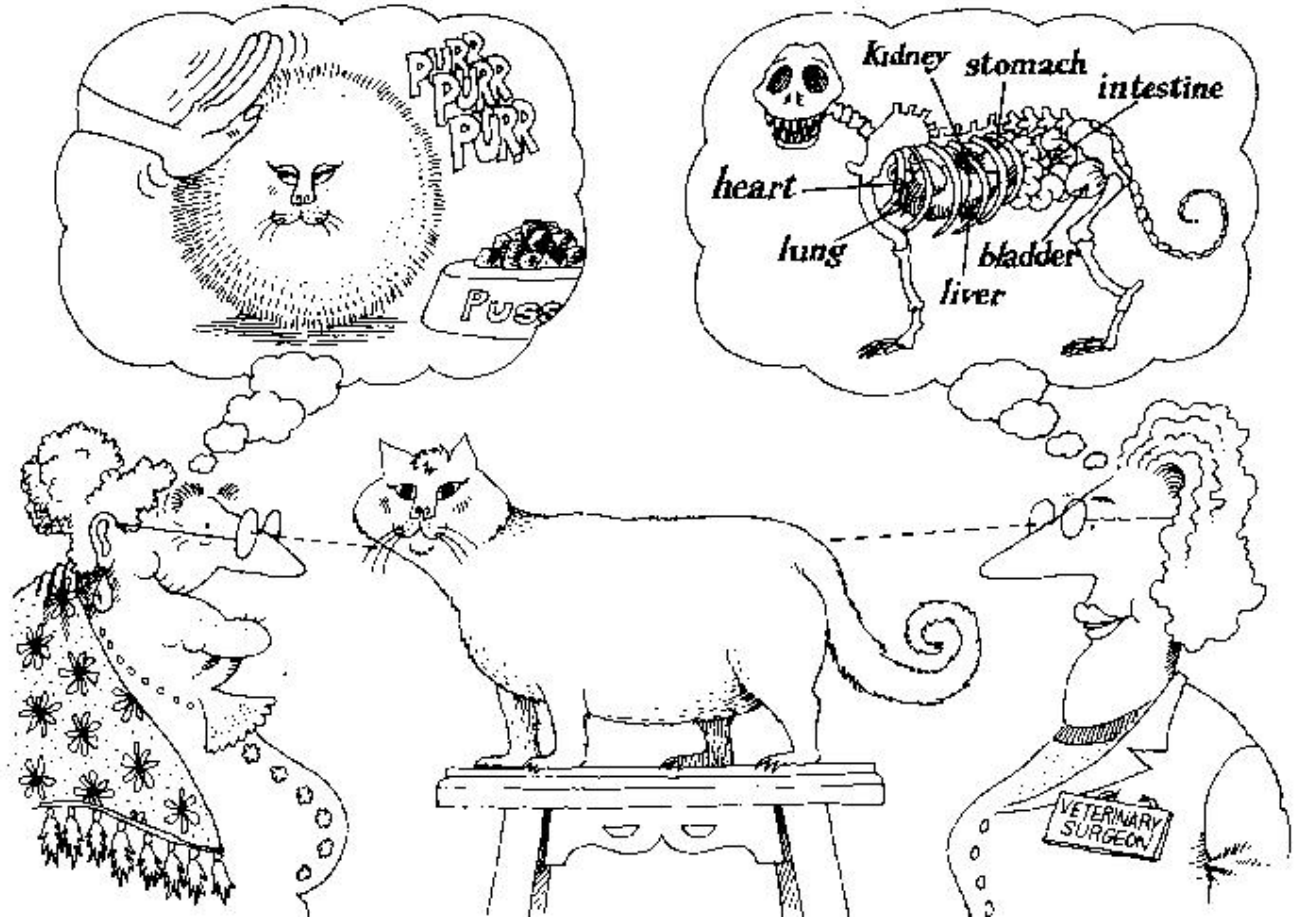
Cosa trascinare via? Un concetto, una idea, un principio da una realtà concreta.

In ambito scientifico, *astrarre significa cambiare la rappresentazione di un problema*.

Perché astrarre?

L'obiettivo del cambio di rappresentazione è quello di concentrarsi su aspetti rilevanti dimenticando gli elementi incidentali.

L'astrazione



L'astrazione si focalizza sulle caratteristiche essenziali di un oggetto, rispetto alla prospettiva di colui che osserva.

Astrazione: processo o entità

Il termine astrazione sotto-intende

- Un processo: l'estrazione delle informazioni essenziali e rilevanti per un particolare scopo, ignorando il resto dell'informazione.
- Una entità: una descrizione semplificata di un sistema che enfatizza alcuni dei dettagli o proprietà trascurandone altri.

Entrambe le viste sono valide e di fatto necessarie.

Astrazione e software

- Similmente, nella **programmazione** l'astrazione allude alla distinzione che si fa tra:
  - cosa (*what*) fa un pezzo di codice
  - come (*how*) esso è implementato

  Per l'utente del codice l'essenziale è cosa fa il codice mentre non è interessato ai dettagli della implementazione.

Che importanza riveste l'astrazione nell'informatica?

Astrazione e software

I sistemi software diventano sempre più complessi.

Per padroneggiare la complessità è necessario concentrarsi solo sui pochi aspetti che più interessano in un certo contesto ed ignorare i restanti.

L'astrazione permette ai progettisti di sistemi software di risolvere problemi complessi in modo organizzato e gestibile.

Astrazione funzionale

L'astrazione funzionale si riferisce alla **progettazione del software**, e in particolare alla possibilità di specificare un modulo software che **trasforma** dei dati di input in dati di output nascondendo i dettagli algoritmici della trasformazione.

- In sintesi:
  - Il modulo software deve trasformare un input in un output, cioè deve calcolare una funzione
  - I dettagli della trasformazione, cioè del calcolo, non sono visibili al consumatore (fruitore) del modulo.
  - Il consumatore conosce solo le corrette convenzioni di chiamata (specifica sintattica) e 'cosa' fa il modulo (specifica semantica).
  - Il consumatore deve fidarsi del risultato.

Astrazione funzionale

Esempio: modulo che realizza un operatore per il calcolo del fattoriale.

La *specifica sintattica* indica il nome del modulo (e.g., **fatt**), il tipo di dato dato in input (un intero) e il tipo di risultato (un intero), in modo da permettere la corretta chiamata del modulo. fatt(intero) → intero

La *specifica semantica* indica la trasformazione operata, cioè la funzione calcolata:

$$fatt(n) = \begin{cases} n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 & n \geq 1 \\ 1 & n = 0 \end{cases}$$

Come la trasformazione è calcolata o realizzata (e.g., iterativamente o ricorsivamente) non è noto al fruitore del modulo.

Astrazione funzionale: specifiche semantiche

Problema: come specificare la semantica del modulo?

Un modo è quello di esprimere, mediante due predicati, la relazione che lega i dati di ingresso ai dati di uscita:

se il primo predicato (detto precondizione) è vero sui dati di ingresso e se il programma termina su quei dati, allora il secondo (detto postcondizione) è vero sui dati di uscita.

Queste specifiche semantiche sono dette assiomatiche.

Nell'esempio del fattoriale:

Precondizione: $n \in N$

Postcondizione: $fatt(n) = \begin{cases} n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 & n \geq 1 \\ 1 & n = 0 \end{cases}$

Astrazione funzionale: specifiche semantiche

Esercizio:

Un modulo ha in input un intero x e un vettore A di n interi e restituisce un intero p e un vettore B di n interi.

Siano date le seguenti:

Precondizione: $\{ n>0 \ \wedge \forall i \in [1,n] \ A[i] \in Z \}$

Postcondizione: $\{ \forall i \in [1,n] \ \exists \ j \in [1,n] \ B[i]=A[j] \ \wedge$
$\forall i \in [1,p-1] \ B[i] \leq x \wedge \forall i \in [p+1,n] \ B[i]>x \ \}$

Riconoscete la funzione di questo modulo?

Limiti dell'astrazione funzionale

L'astrazione funzionale non permette di progettare, e quindi sviluppare, moduli software invarianti ai cambiamenti nei dati (sono invarianti solo ai cambiamenti nei processi di trasformazione che operano).

Questo rende difficoltosa la manutenzione delle soluzioni progettate.

È inappropriata per lo sviluppo di soluzioni a problemi complessi.

Astrazione dati

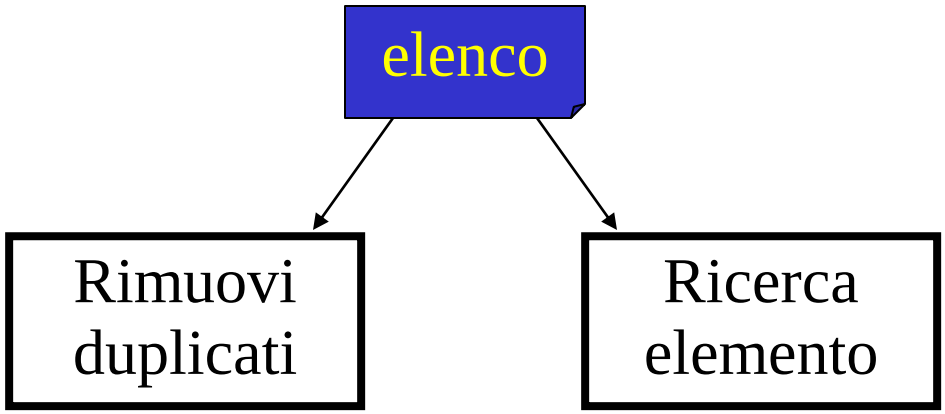
Alla base dell'**astrazione dati** c'è il principio che *non si può accedere direttamente alla rappresentazione di un dato, qualunque esso sia, ma solo attraverso un insieme di operazioni considerate lecite*

(principio dell'astrazione dati).

VANTAGGIO: un cambiamento nella rappresentazione del dato si ripercuoterà solo sulle operazioni lecite, che potrebbero subire delle modifiche, mentre non inficerà il codice che utilizza il dato astratto.

Astrazione dati

Esempio: Se i moduli ‘Rimuovi duplicati’ e ‘Ricerca elemento’ accedono all’elenco attraverso un insieme si operazioni lecite (e.g., ‘dammi il prossimo elemento’, ‘non ci sono più elementi’, ecc.) il cambiamento della rappresentazione dell’elenco richiederà una riformulazione delle operazioni lecite, ma non influenzerà i due moduli.



Information Hiding

In generale un principio di astrazione suggerisce di occultare l'informazione (*information hiding*) sulla rappresentazione del dato

- sia perché non necessaria al fruitore dell'entità astratta
- sia perché la sua rivelazione creerebbe delle inutili dipendenze che comprometterebbero l'invarianza ai cambiamenti.

Information Hiding

Il principio dell'astrazione funzionale suggerisce di occultare i dettagli del processo di trasformazione ("come" esso è operato).

Il principio dell'astrazione dati identifica nella rappresentazione del dato l'informazione da nascondere.

In entrambi i casi non si dice COME farlo.

Questo sarà chiarito quando approfondiremo il tema dell'astrazione nella programmazione.

Incapsulamento

L'incapsulamento (encapsulation) è una tecnica di progettazione consistente nell'impacchettare (o "racchiudere in capsule") una collezione di entità, creandone una barriera concettuale.

Come l'astrazione, l'incapsulamento sottointende

- *Un processo*: l'impacchettamento
- *Una entità*: il 'pacchetto' ottenuto

Ad essa corrispondono tecniche di programmazione che consentono l'incapsulamento dei dati.

Incapsulamento

Esempi:

- una procedura impachetta diversi comandi
- una libreria incapsula diverse funzioni
- un oggetto incapsula un dato e un insieme di operazioni sul dato

Incapsulamento

L'incapsulamento NON dice come devono essere le "pareti" del pacchetto (o capsula), che potranno essere:

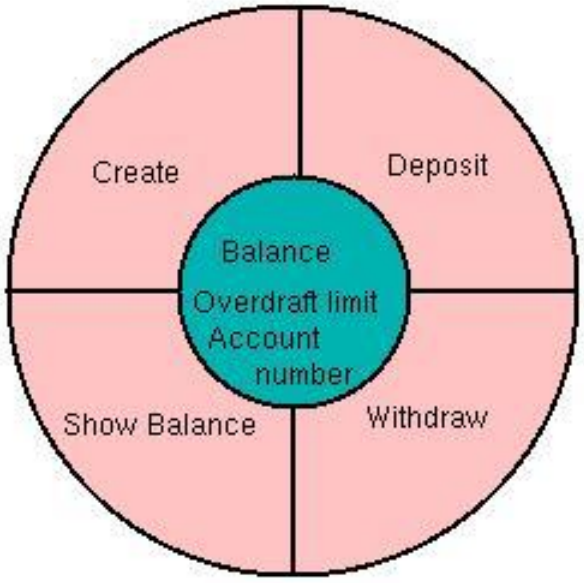
- Trasparenti: permettendo di vedere tutto ciò che è stato impacchettato;
- Traslucide: permettendo di vedere in modo parziale il contenuto;
- Opache: nascondendo tutto il contenuto del pacchetto.

Astrazione dati & Incapsulamento

La combinazione del principio dell'astrazione dati con la tecnica dell'incapsulamento suggerisce che:

1. La rappresentazione del dato va nascosta
2. L'accesso al dato deve passare solo attraverso operazioni lecite
3. Le operazioni lecite, che ovviamente devono avere accesso alla informazione sulla rappresentazione del dato, vanno impacchettate con la rappresentazione del dato stesso.

Astrazione dati & Incapsulamento



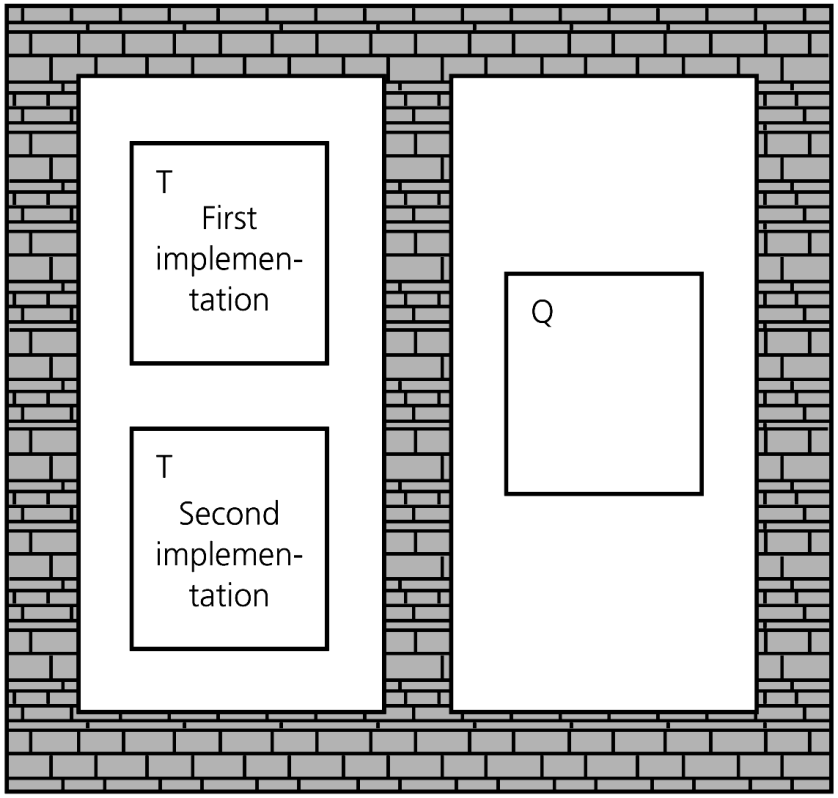
Esempio: il dato "conto corrente" ha una sua rappresentazione interna che permette di memorizzare il saldo (*balance*), il limite fido (*overcraft limit*) e il numero di conto (*account number*). La rappresentazione interna dei tre dati è nascosta. L'accesso alla rappresentazione passa per tre operazioni lecite:

- Creazione conto
- Deposito
- Prelievo
- Stampa saldo

Rappresentazione e operazioni lecite sono impacchettate in un modulo.

Astrazione dati & Incapsulamento

T e Q sono isolati: l'implementazione di T non influenza Q



Astrazione dati vs. Astrazione funzionale

L'astrazione dati ricalca ed estende quella funzionale.

Attualmente la possibilità di effettuare astrazioni di dati è considerata importante almeno quanto quella di definire nuovi operatori con astrazioni funzionali. L'esperienza ha infatti dimostrato che la scelta delle strutture di dati è il primo passo sostanziale per un buon risultato dell'attività di programmazione.

L'astrazione funzionale stimola gli sforzi per evidenziare operazioni ricorrenti o comunque ben caratterizzate all'interno della soluzione di un dato problema.

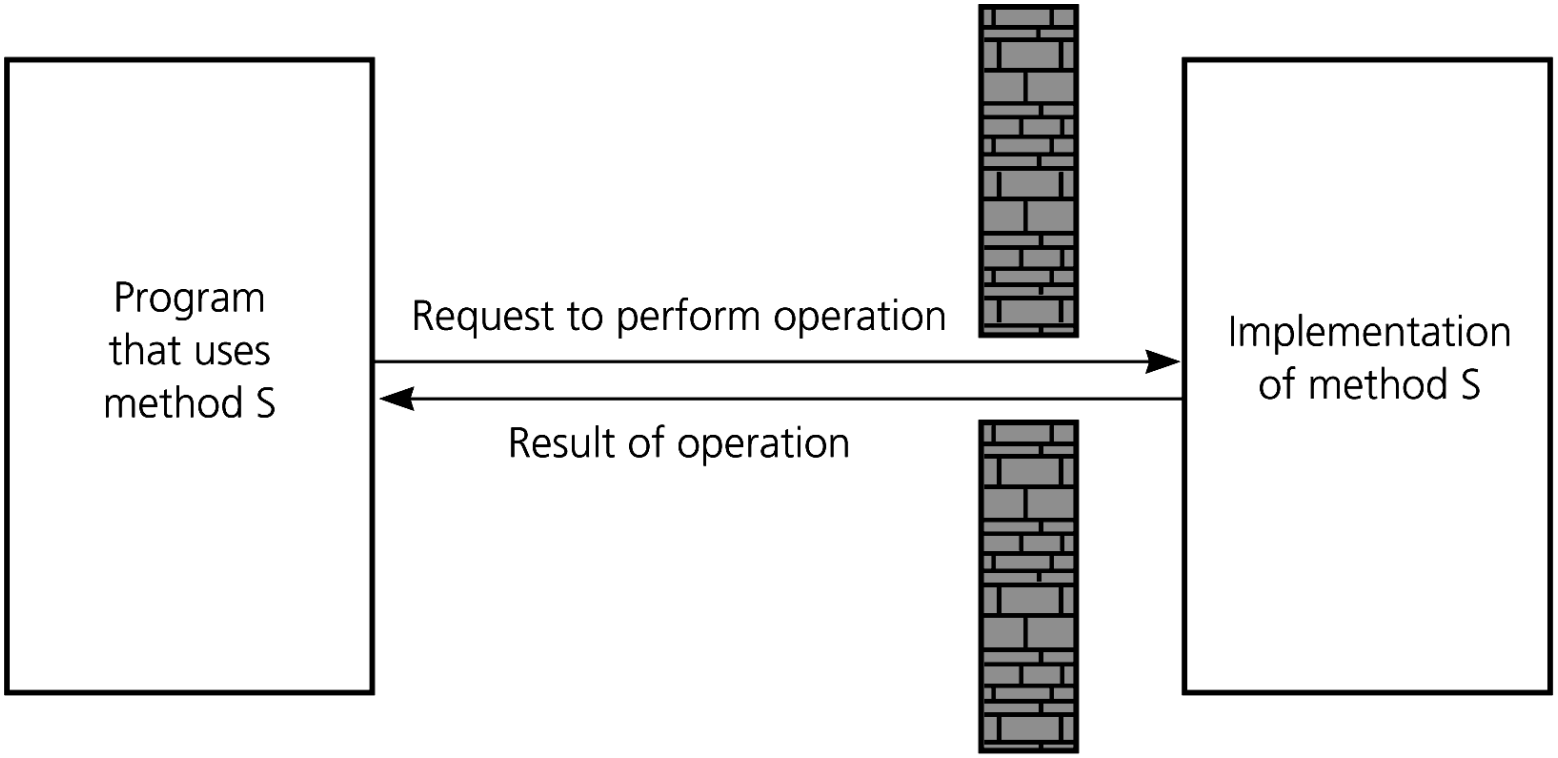
L'astrazione di dati stimola in più gli sforzi per individuare le organizzazione dei dati più consone alla soluzione del problema.

Da una progettazione *function centered* a una *data centered*.

Astrazione dati & Incapsulamento

Ovviamente, l'isolamento dei moduli non può essere totale.

- La **specifica**, o **contratto**, descrive come si può interagire con un dato astratto.



Astrazione: i punti di vista

In generale, le astrazioni supportano la separazione dei diversi interessi di

- **Utenti**: interessati a cosa si astrae (*what*)
- **Implementatori**: interessati a come (*how*) si realizza.

Per questa ragione una definizione di astrazione ha sempre due componenti:

- **Specifica**
- **Realizzazione**

Per descrivere una specifica occorre ricorrere a dei *linguaggi di specifica*, che sono diversi dai linguaggi usati per descrivere le realizzazione delle astrazioni.

Specifica sintattica e semantica

La specifica potrà poi essere:

- **Sintattica**: stabilisce quali identificatori sono associati all'astrazione
- **Semantica**: definisce il risultato della computazione inclusa nell'astrazione.

Parametrizzazione di un'astrazione

L'efficacia di un'astrazione può essere migliorata mediante l'uso di **parametri** per la comunicazione con l'ambiente esterno.

Quando un'astrazione è chiamata, ciascun parametro formale verrà associato in qualche modo al corrispondente argomento.

Astrazione dati

Anche un'astrazione dati, come una qualunque astrazione, è costituita da una ***specifica*** e una ***realizzazione***:

- La *specifica* consente di descrivere un nuovo dato e gli operatori che ad esso sono applicabili.
- La *realizzazione* stabilisce come il nuovo dato e i nuovi operatori vengono ricondotti ai dati e agli operatori già disponibili.

Chi intende usare i nuovi dati con i nuovi operatori nella scrittura dei suoi programmi sarà tenuto a conoscere la specifica dell'astrazione dei dati, ma potrà astrarre dalle tecniche utilizzate per la realizzazione.

Astrazione dati

I linguaggi di specifica per astrazione dati più noti sono due:

- Il linguaggio logico-matematico usato nelle asserzioni ➔ specifiche assiomatiche
- Il linguaggio dell'algebra usato nelle equazioni definite fra gli operatori specificati nel dato astratto ➔specifiche algebriche.

Astrazione dati: le specifiche assiomatiche

Un linguaggio formale per la specifica di un tipo astratto di dato è costituito dalla notazione logico-matematica delle ***asserzioni***. In questo caso si parla di specifica assiomatica.

Una specifica assiomatica consta di:

a) Una specifica ***sintattica*** (detta segnatura, *signature)* che fornisce:

a**1**) l'elenco dei NOMI dei domini e delle operazioni specifiche del tipo;

a**2**) i DOMINI di partenza e di arrivo per ogni nome di operatore;

Astrazione dati: le specifiche assiomatiche

b) Una specifica ***semantica*** che associa:

b**1**) un INSIEME ad ogni nome di tipo introdotto nella specifica sintattica;

b**2**) una FUNZIONE ad ogni nome di operatore, esplicitando le seguenti condizioni sui domini di partenza e di arrivo:

i) *precondizione*, che definisce quando l'operatore è applicabile

ii) *postcondizione*, che stabilisce la relazione tra argomenti e risultato

Il dato astratto *Vettore*

Un esempio elementare di specifica assiomatica è quella di un vettore

a) Specifica sintattica

Tipi: vettore, intero, tipoelem

Operatori:

CREAVETTORE() → vettore

LEGGIVETTORE(vettore, intero) → tipoelem

SCRIVIVETTORE(vettore, intero, tipoelem) → vettore

Il dato astratto *Vettore*

b) *Specifica semantica*

Tipi:

intero: l'insieme dei numeri interi

vettore: l'insieme delle sequenze di *n* elementi di tipo *tipoelem*

Operatori:

CREAVETTORE = v

Pre: *non ci sono precondizioni, o più precisamente, il predicato che definisce la precondizione di applicabilità di CREAVETTORE è il valore VERO*

Post: $\forall i \in \{0,1, 2, \ldots, n-1\}$, l'*i*-esimo elemento del vettore, *v(i)*, è uguale ad un prefissato elemento di tipo tipoelem

LEGGIVETTORE(v,i) = e

Pre: $0 \leq i \leq n-1$

Post: e=v(i)

SCRIVIVETTORE(v,i,e) = v'

Pre: $0 \leq i \leq n-1$

Post: $\forall j \in \{0,1, \ldots, n-1\}$, $j \neq i$, v'(j)=v(j), v'(i)=e

Il dato astratto *Vettore*

c) Realizzazione

In Java il vettore (array) è un tipo di dato concreto. La corrispondenza tra la specifica appena introdotta e quella del Java è la seguente:

CREAVETTORE	⇔	tipoelem v[n]
LEGGIVETTORE(v,i)	⇔	v[i]
SCRIVIVETTORE(v,i,e)	⇔	v[i] = e

dove *i* può essere anche una espressione di tipo intero.

Limiti delle specifiche assiomatiche

Si noti che il metodo di specifica assiomatica è preciso nella definizione della specifica sintattica, mentre è piuttosto informale per gli altri aspetti (talvolta si ricorre al linguaggio naturale per semplicità). Pertanto esso non consente di caratterizzare precisamente un tipo astratto.

In particolare non consente di definire i valori che possono essere generati mediante l'applicazione di operatori, e non consente di stabilire quando l'applicazione di diverse sequenze di operatori porta al medesimo valore.

Questo problema è superato dalle specifiche algebriche.

Dato Astratto: specifiche algebriche

Le specifiche algebriche si basano sull'algebra, piuttosto che sulla logica. Essenzialmente definiscono un dato astratto come un'algebra eterogenea, ovvero come una collezione di diversi insiemi su cui sono definite diverse operazioni.

Le algebre tradizionali sono omogenee. Un'algebra omogenea consiste in un unico insieme e diverse operazioni.

Esempio: Z con le operazioni di addizione e moltiplicazione è un algebra omogenea.

Dato Astratto: specifiche algebriche

Esempio: dato un alfabeto Σ, indichiamo con Σ^* l'insieme di tutte le stringhe, incluso quella vuota, costruite con i simboli di Σ.

Σ^* con le operazioni di concatenazione e calcolo della lunghezza non sono un'algebra omogenea, visto che il codominio dell'operazione di calcolo della lunghezza è N e non Σ^*.

Quindi quest'algebra consiste di due insiemi, stringhe e interi, su cui sono definite le operazioni di concatenamento e lunghezza delle stringhe.

ALGEBRA ETEROGENEA

Dato Astratto: specifiche algebriche

Una specifica algebrica consiste di tre parti:

1. ***Sintattica:*** elenca i nomi del tipo, le sue operazioni e il tipo degli argomenti delle operazioni. Se un'operazione è una funzione allora è specificato il codominio (range) della funzione. → DIVERSA DA QUELLA CHE GIÀ CONOSCIAMO
2. ***Semantica:*** consiste di un insieme di equazioni algebriche che descrivono in modo indipendente dalla rappresentazione le proprietà delle operazioni.
3. ***Di restrizione:*** stabilisce varie condizioni che devono essere soddisfatte o prima che siano applicate le operazioni o dopo che esse siano state completate.

N.B.: alcuni autori inglobano le specifiche di restrizione in quelle semantiche.

Dato Astratto: specifiche algebriche

Uno degli aspetti più interessanti delle specifiche algebriche è la semplicità del linguaggio di specifica rispetto ai linguaggi di programmazione procedurale.

Infatti, il linguaggio di specifica consiste di solo cinque primitive:

1. composizione funzionale $f: A \to B \quad g: C \to D \to f \circ g$
2. relazione di eguaglianza
3. 4. due costanti, true e false
5. un numero illimitato di variabili libere

A DIFFERENZA DELLA SPECIFICA SEMANTICA LIE COSTRINGE LE VAR AI QUANTIFICATORI

Dato Astratto: specifiche algebriche

MATEMATICA : DOMINIO È IL PRODOTTO CARTESIANO FRA UN BOOLEAN E DUE STATEMENT (COMP. DI FUNZIONI). IL CODOMINIO È ANCH'ESSO UNO STATEMENT

La funzione *if then else* può essere facilmente descritta dalle seguenti equazioni:

if then else (true,q,r) = q
if then else (false,q,r) = r

Questa funzione è così importante che si assume già data come operatore infisso:

if p then q else r

Inoltre si assume che sono predefiniti i valori interi e booleani.

Specifica algebrica di *Pila*

STRUTTURA DI TIPO LIFO

Specifica sintattica

sorts: stack, item, boolean

operations:

newstack() → stack

push(stack, item) → stack

pop(stack) → stack

top(stack) → item

isnew(stack) → boolean

Ogni insieme è detto un sort → DOMINIO (letteralmente "tipo") dell'algebra eterogenea.

I *sort* **item** e **boolean** sono ausiliari alla definizione di stack.

Specifica algebrica di *Pila*

Specifica semantica

declare stk: stack, i: item

pop(push(stk, i)) = stk

top(push(stk, i)) = i

isnew(newstack) = true

isnew(push(stk, i)) = false

Nella specifica semantica occorre dichiarare i parametri su cui lavorano gli operatori.

→ ISNEW (NEWSTACK()) = TRUE

Specifica algebrica di *Pila*

Specifica di restrizione

restrictions

pop(newstack) = error

top(newstack) = error

dove ‘error’ è un elemento speciale indefinito.

Specifica algebrica di *Pila*

Ovviamente avremmo potuto scrivere molte altre equazioni, come:

$$isnew(pop(push(newstack, i))) = true$$

che evidenzia come il predicato *isnew* sia vero anche per quegli stack ottenuti inserendo un generico elemento *i* in un **nuovo** stack e poi rimuovendolo.

Tuttavia questa equazione è ridondante, poiché è ricavabile dalle altre scritte in precedenza. Infatti, grazie alla prima e quarta equazione possiamo scrivere:

isnew(pop(push(newstack, i))) = isnew(newstack)=true

Specifica algebrica di *Pila*

Nelle specifiche semantiche è importante indicare l'insieme minimale di equazioni (dette assiomi) a partire dalle quali possiamo derivare tutte le altre.

Le specifiche semantiche si diranno incomplete se non permetteranno di derivare tutte le verità (equazioni) desiderate dell'algebra specificate.

Le specifiche semantiche si diranno inconsistenti (o contraddittorie) se permetteranno di derivare delle equazioni indesiderate, cioè considerate false.

Le specifiche semantiche si diranno ridondanti se alcune delle equazioni sono ricavabili dalle altre.

Costruttori e Osservazioni

Scrivere delle specifiche semantiche complete, consistenti e non ridondanti può non essere un compito semplice.

Per questo conviene introdurre una metodologia, che si basa sulla distinzione degli operatori di un dato astratto in:

- Costruttori, che creano o istanziano il dato astratto
- Osservazioni, che ritrovano informazioni sul dato astratto

Il comportamento di una astrazione dati può essere specificata riportando il valore di ciascuna osservazione applicata a ciascun costruttore.

Questa informazione è organizzata in modo naturale in una matrice, con i costruttori lungo una dimensione e le osservazioni lungo l'altra.

Costruttori e Osservazioni

osservazioni	***Costruttore di stk'***	
	newstack	*push(stk, i)*
pop(stk')	error	stk
top(stk')	error	i
isnew(stk')	true	false

Osservazioni binarie

Tutte le osservazioni viste finora sono unarie, nel senso che esse osservano un singolo valore del dato astratto.

Spesso è necessario disporre di osservazioni più complesse.

Per confrontare due valori è necessario osservare due istanze del dato astratto.

Questo complica la specifica, perché il valore dell'osservazione dev'essere definito per tutte le combinazioni di costruttori possibili per i valori astratti che si devono confrontare.

Osservazione binaria: equal(l,m)

Esempio: predicato *equal(l,m)* che è vero solo se le due pile contengono gli stessi elementi nello stesso ordine.

Costruttore di m	***Costruttore di l***	
	newstack	*push(stk, i)*
newstack	true	false
push(stk', i')	false	i=i' and ∧ equal(stk,stk')

Questa tabella può essere vista come l'aggiunta di una terza dimensione alla tabella di base per le osservazioni unarie.

ASSUMO ∧ : BOOLEAN × BOOLEAN = BOOLEAN

Osservazione binaria: equal(stk',m)

Questa terza dimensione può essere rimossa andando a classificare esplicitamente solo il primo argomento dell'osservazione e referenziando in modo astratto il secondo argomento mediante una variabile libera.

osservazione	***Costruttore di stk'***	
	newstack	*push(stk, i)*
equal(stk',m)	isnew(m)	* **not** isnew(m) **and** i=top(m) **and** equal(stk,pop(m))

* SI ASSUME COME PER AND

Specifica algebrica di coda

Specifica sintattica

sorts: queue, item, boolean

operations:

newq() → queue

addq(queue, item) → queue

deleteq(queue) → queue

frontq(queue) → item

isnewq(queue) → boolean

Costruttori e Osservazioni

osservazioni	***Costruttore di q'***	
	newq	*addq(q, i)*
isnew(q')	true	false
frontq(q')	error	**if** isnewq(q) **then** i **else** frontq(q)
deleteq(q')	error	**if** isnewq(q) **then** newq **else** addq(deleteq(q),i)

Specifica algebrica di coda

Specifica semantica

declare q: queue, i: item

isnewq(newq) = true

isnewq(addq(q, i)) = false

deleteq(addq(q, i)) = **if** isnewq(q) **then** newq **else** addq(deleteq(q),i)

frontq(addq(q, i)) = **if** isnewq(q) **then** i **else** frontq(q)

Specifica algebrica di coda

Specifica di restrizione

restrictions

frontq(newq) = error

deleteq(newq) = error

Esercizi

- Fornire una specifica algebrica semantica completa, consistente e minimale per il dato astratto *Stringa* supposto che le specifiche sintattiche siano le seguenti:

  sorts: string, char, integer, boolean

  operations:

new() → string	crea nuove stringhe
append(string, string) → string	concatena due stringhe
add(string, char) → string	aggiunge un carattere a fine stringa
length(string) → integer	calcola la lunghezza di una stringa
isEmpty(string) → boolean	prédica se la stringa è vuota
equal(string, string) → boolean	prédica se due stringhe sono uguali

La scelta dei costruttori

Talvolta, l'applicazione della metodologia per la sintesi di specifiche algebriche può comportare delle difficoltà riguardo al discernimento di cosa è costruttore e cosa operazione.

Ad esempio, guardando la specifica sintattica del dato astratto *Stringa* possiamo dire che tutti gli operatori che restituiscono una stringa sono dei candidati a essere dei costruttori.

new() → string

append(string, string) → string

add(string, char) → string

Mentre non ci sono dubbi sul primo, restano delle perplessità sul fatto che entrambi gli altri operatori debbano essere dei costruttori.

La scelta dei costruttori

Ritroveremo questa problematica anche nella progettazione orientata a oggetti: cosa dev'essere un costruttore e cosa un metodo?

Nello scegliere i costruttori adotteremo il criterio di minimalità, cioè l'insieme dei costruttori dev'essere il più piccolo insieme di operatori necessario a costruire tutti i possibili valori per un certo dato astratto.

Ora, è evidente che per costruire una stringa abbiamo bisogno di:

new() → string

add(string, char) → string

Il terzo operatore:

append(string, string) → string

non è necessario, quindi non lo scegliamo come costruttore.

Un criterio euristico: *gli argomenti di un costruttore non devono essere tutti dello stesso sort del dato astratto*.

new() → string
append(string, string) → string
add(string, char) → string
length(string) → integer
isEmpty(string) → boolean
equal(string, string) → boolean

Declare : s,s':string; c,c':char

Length(new())=0

Length(add(s,c))=length(s)+1

Assumo +(Integer,Integer)→Integer

isEmpty(new())=true

isEmpty(add(s,c))=false

Equal(new,new)=true

Equal(new, add(s',c'))=false

Equal(add(s,c),new)=false

Equal(add(s,c),add(s',c'))=c=c' and equal (s,s')

Assumo and(Boolean, Boolean)→Boolean

Assumo =(Character, Character)→Boolean

Append(new, new)=new()

Append(new, add(s',c'))=add(s',c')

append(add(s,c), add(s',c')) = add(append(add(s,c), s'), c')

Esercizio (in aula)

Progettare il dato astratto *Conto Con Fido* (account with overcraft) che consente rappresentare dei conti correnti bancari per i quali è permesso uno scoperto. Si deve poter limitare lo scoperto tramite concessione del fido. Le operazioni ammesse per questo dato astratto sono:

conto (account): apre un nuovo conto corrente bancario definendo saldo iniziale e massimo scoperto ammesso

saldo (balance): riporta il saldo del conto

deposita (deposit): deposita una somma sul conto

preleva (withdraw): preleva denaro dal conto

concediFido (setOverdraftLimit): definisce il limite massimo dello scoperto

fidoConcesso (getOverdraftLimit): restituisce il limite massimo dello scoperto

Esercizio (cont.)

Specifica sintattica

sorts contocorrente, saldo, denaro, scoperto;

conto(saldo, scoperto)→contocorrente

saldo(contocorrente) → saldo

deposita(contocorrente, denaro) → contocorrente

preleva(contocorrente, denaro) → contocorrente

concediFido(contocorrente, scoperto) → contocorrente

fidoConcesso(contocorrente) → scoperto

Esercizio (cont.)

Specifica semantica

Il dominio *contocorrente* è l'insieme delle coppie $(s,l) \in \mathbf{R} \times \mathbf{R}^+$, tali che $s \geq -l$. Il dominio *saldo* corrisponde all'insieme dei numeri reali **R**, mentre i domini *scoperto* e *denaro* corrispondono ad $\mathbf{R}^+$.

Pertanto potremo disporre sui vari domini di tipiche operazioni algebriche, e in particolare:

- due operazioni binarie: "+" e "-"
- operatore unario: "-"

nonché di una relazione binaria "<" e di una costante "0".

N.B.: formalmente avremmo dovuto indicare che "+" è definito sui domini *saldo* e *scoperto* e restituisce un *saldo*, etc.,

Esercizio (cont.)

N.B.: Gli operatori aggiuntivi che sono stati specificati non fanno riferimento al dominio *contocorrente* relativo al dato astratto che si vuole definire. Per questo non è necessario specificarli.

Diversamente, se fosse stata ipotizzata la disponibilità di un operatore del dominio *contocorrente*, avremmo dovuto successivamente indicarne la semantica al pari degli altri operatori indicati nella specifica sintattica.

Esercizio (cont.)

Possiamo identificare un solo costruttore (*conto*). Gli altri operatori sono tutte osservazioni.

osservazioni	***Costruttore di c***
	conto(s,l)
saldo(c)	s
deposita(c,d)	conto(s+d,l)
preleva(c,d)	**if** s-d<-l **then** error **else** conto(s-d,l)
concediFido(c,l')	**if** s<-l' **then** error **else** conto(s,l')
fidoConcesso(c)	L

Assumo +(Saldo, Denaro)→Saldo – (Saldo, Denaro)→Saldo – (Scoperto)→Scoperto, <(Saldo, Scoperto) →Boolean

Declare s: Saldo, l, l':scoperto d:denaro

Esercizio (cont.)

Specifica semantica

declare s: saldo, d: denaro, l, l': scoperto;

saldo(conto(s,l))=s

deposita(conto(s,l), d) = conto(s+d,l)

preleva(conto(s,l), d) = **if** s-d<-l **then** error **else** conto(s-d,l)

concediFido(conto(s,l),l') = **if** s<-l' **then** error **else** conto(s,l')

fidoConcesso(conto(s,l))=l

Esercizio (cont.)

Utilizzando le precedenti specifiche è possibile dimostrare la seguente equazione:

si aggiunge declare c:conto

preleva(deposita(c,d),d)=c

Dim.: Sia c=conto(s,l)

preleva(deposita(conto(s,l),d),d)=preleva(conto(s+d,l), d)

per definizione di *deposita* (deposita(conto(s,l),d)=conto(s+d,l)).

Inoltre:

preleva(conto(s+d,l), d)=conto((s+d)-d,l)

per definizione di *preleva* (preleva(conto(s,l), d) = if s-d<-l then error else conto(s-d,l)).

Ma conto((s+d)-d,l)=conto(s,l)=c

c.v.d.

Esempio: dato astratto Dizionario

Progettare il dato astratto *Dizionario* che consente di gestire un un in insieme di coppie (chiave e valore). L'insieme di operazioni ammesse è il seguente:

creaDizionario: restituisce un dizionario vuoto

aggiungiCoppia: aggiunge una coppia composta da chiave e valore al dizionario

cancella: cancella tutte le occorrenze di un dato valore , restituisce errore in assenza di occorrenze da rimuovere

Dato astratto Dizionario

Specifica sintattica

sorts : Dizionario, Chiave, Valore, Integer, Boolean

- creaDizionario()→Dizionario
- aggiungiCoppia(Dizionario, Chiave,Valore)→Dizionario
- cancella(Dizionario,Valore)→Dizionario

Aggiungi specifica sintattica di "conta" (operatore privato)

Dizionario (spec. semantica)

Specifica semantica

declare d: Dizionario; k: Chiave; v,v': Valore;

osservazioni	***Costruttore di d'***	
	creaDizionario()	*aggiungiCoppia(d,k,v)*
cancella(d',v')	error	if(v=v') then if conta(d,v')>=1 then cancella(d,v') else d else aggiungiCoppia(cancella(d,v'),k,v)
private: *conta(d',v')*	0	if(v=v') then conta(d,v')+1 else conta(d,v')

Assumo operatore +:InteroxIntero --> Intero

Riferimenti bibliografici

Per la progettazione con astrazione funzionale e astrazione dati

Peter Klein

Designing Software with Modula-3

Per le specifiche assiomatiche di vettore, pila e coda

Alan Bertossi

Algoritmi e Strutture di Dati

UTET, 2000

Capitoli 0, 1, 4

Per le specifiche algebriche

John D. Gannon, James M. Purtilo, Marvin V. Zelkowitz

Software Specification: A Comparison of Formal Methods

2001

Capitolo 5.3

Riferimenti bibliografici

Per le specifiche algebriche si può fare riferimento anche al testo:

A. Fuggetta, C. Ghezzi, S. Morasca, A. Morzenti, M. Pezzè

Ingegneria del software: progettazione, sviluppo e verifica

Mondadori Informatica, 1991

Capitolo 4.3