Astrazione nella progettazione

Programmazione II corso A
Corso di Laurea in ITPS
Università degli studi di Bari Aldo Moro
a.a. 2022-2023

Indice

- Astrazione
  - Astrazione funzionale
  - Astrazione dati
    - Information Hiding
    - Incapsulamento
    - Astrazione dati ed incapsulamento
    - Specifiche assiomatiche nell'Astrazione dati
  - Astrazione di controllo

Astrazione: definizione ed obiettivo

- Astrarre: dal lat. *abstrahere* = 'trascinare via', dal verbo *trahere* = trascinare.
  - Cosa trascinare via? Un concetto, una idea, un principio da una realtà concreta.

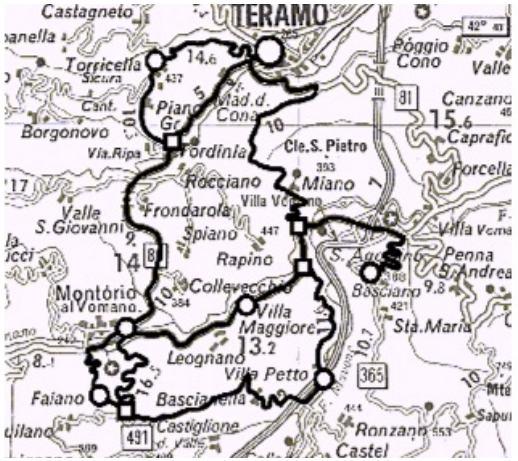
In ambito scientifico, *astrarre significa cambiare la rappresentazione di un problema*

Astrazione: obiettivo

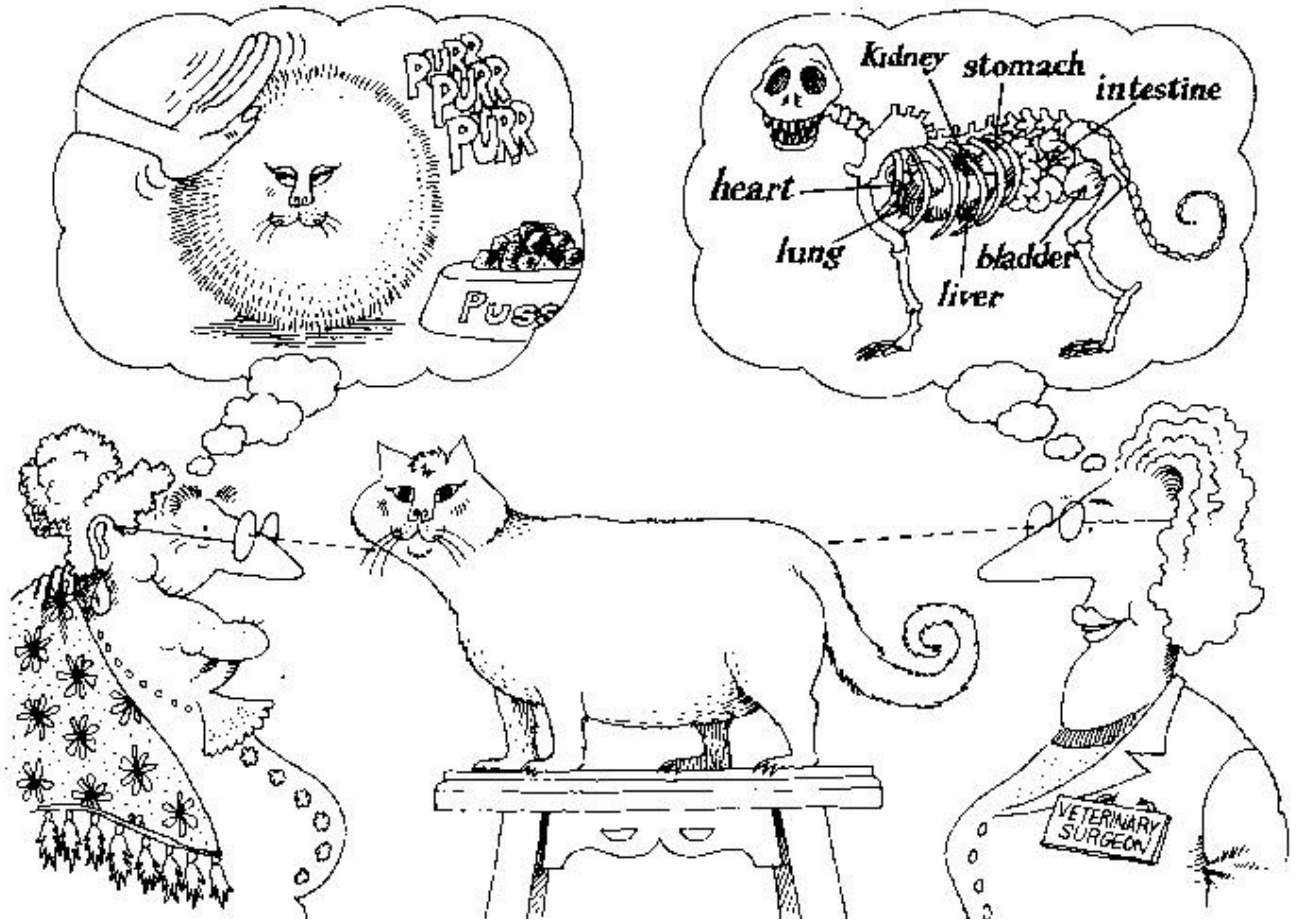
- Concentrarsi su aspetti rilevanti dimenticando gli elementi incidentali.
  - non si tratta di omettere parti della rappresentazione di un problema, ma di riformulare lo stesso concentrando l'attenzione su idee generali piuttosto che su manifestazioni specifiche di quelle idee, tenendo conto della **prospettiva** di un osservatore.

Esempio: trovare il percorso migliore per un commesso viaggiatore che deve visitare diverse città.

Se 'migliore' significasse 'più breve', potremmo trascurare fattori come presenza di traffico e qualità della viabilità, e riformulare il problema come ricerca di minimo percorso in un grafo.



L'astrazione: esempio



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 e con una particolare prospettiva, 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.

L'astrazione nell'analisi dei sistemi

- Nell'**analisi dei sistemi** un'astrazione è una descrizione semplificata del sistema, che enfatizza alcuni dettagli o proprietà essenziali del sistema mentre ne ignora altri estranei.

Esempio: Sistema di controllo del traffico aereo

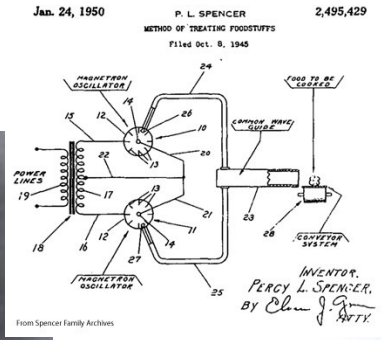
–Essenziale: posizione del velivolo, velocità, …

–Irrilevante: colore, nomi dei passeggeri, etc.

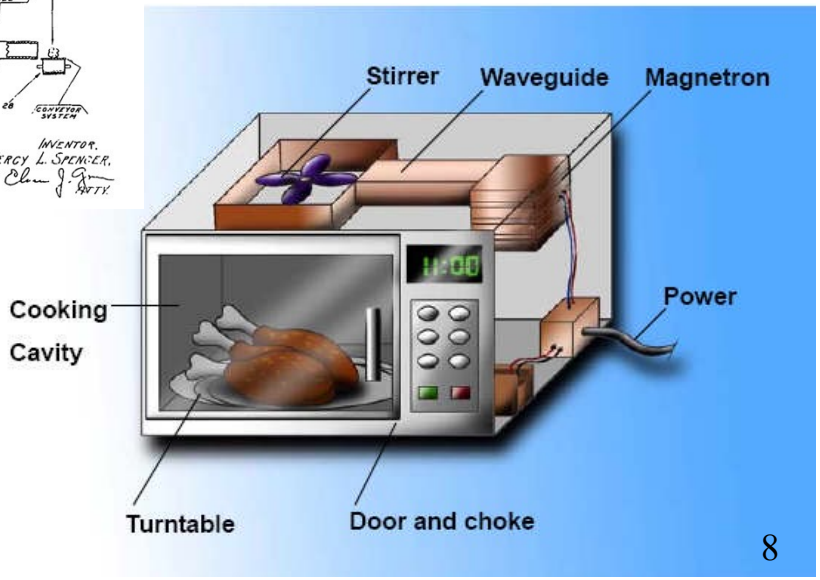
L'astrazione nell'analisi dei sistemi

Nel quotidiano il principio di astrazione è costantemente applicato ogni qualvolta utilizziamo uno strumento senza per questo sapere come è realizzato.

Esempio: utilizzo di un forno a microonde senza conoscerne la realizzazione.



brevetto



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.
    - Domanda: quali sono esempi di aspetti di interesse?
  - L'astrazione permette ai progettisti di sistemi software di risolvere problemi complessi in modo organizzato e gestibile.
    - Domanda: quali sono esempi di astrazione nella produzione del software?

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 fornito 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: affermazione nell'informatica

- L'astrazione funzionale si è affermata pienamente solo nei primi anni '70, quando emerse una metodologia che mirava a costruire i programmi progredendo dal generale al particolare ("**stepwise refinement**" o "**top-down programming**").
  - Secondo questa metodologia, per risolvere un problema (o affrontare un compito) P si procede come segue:

Metodologia *Stepwise refinement*

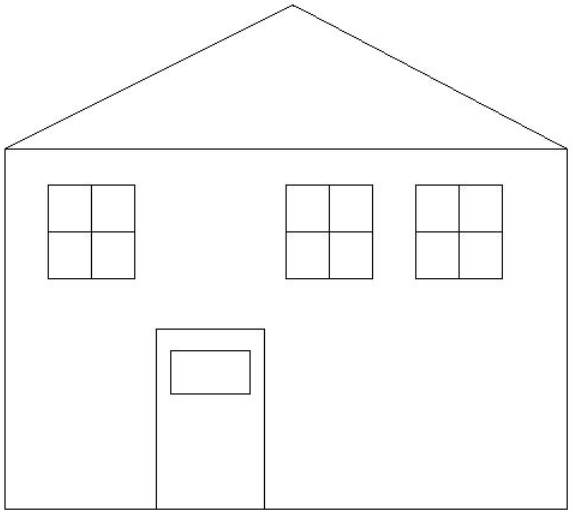
1. Decomponi il compito P in sottocompiti P_1, P_2, …, P_n
2. Ipotizza di disporre di moduli M_1, M_2, … M_n che effettuano le traformazioni richieste rispettivamente da P_1, P_2, …, P_n
3. Componi un modulo M che assolve al compito P usando i moduli M_1, M_2, … M_n
4. Applica ricorsivamente la metodologia ai sottocompiti P_1, P_2, …, P_n al fine di definire la realizzazione di M_1, M_2, … M_n fino a quando non si ottengono sottocompiti considerati elementari (o non ulteriormente decomponibili).

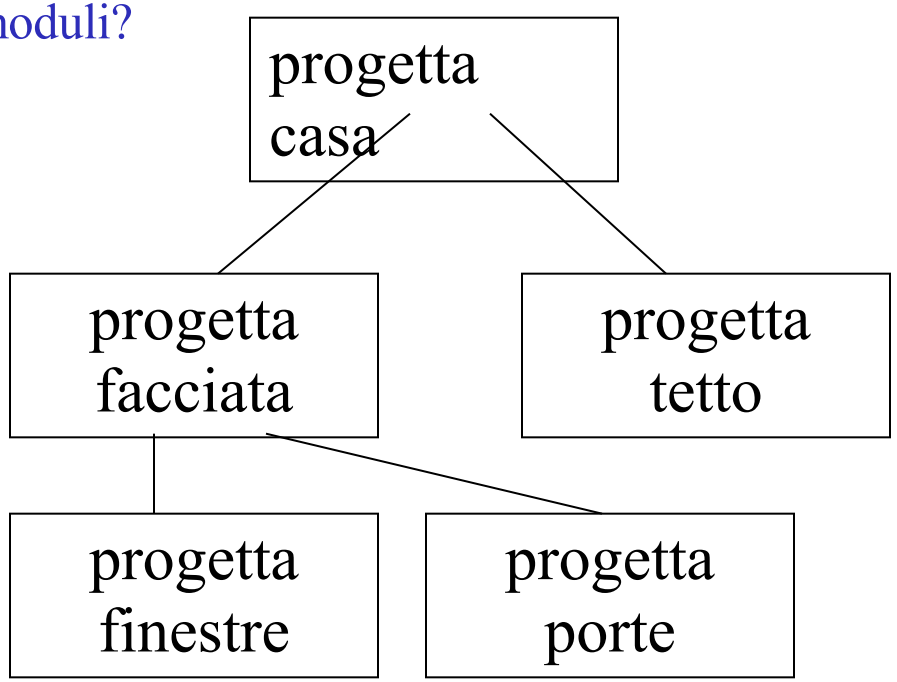
Esempio di *Stepwise refinement*

Tipicamente la dipendenza fra moduli è rappresentata da un albero.

Domanda: qual è il modulo M che risolve il problema?

Domanda: cosa implica la presenza dei moduli?





Metodologia *Stepwise refinement*

- il programmatore è libero di assumere l'esistenza di qualsiasi modulo (detto *stub*, lett. *matrice* di qualcosa, e.g., assegno) che si può applicare al particolare sottocompito e di cui fornisce una specifica, salvo dover poi specificare come quel modulo va realizzato.

NOTA BENE

L'astrazione funzionale è di supporto alla metodologia dello stepwise refinement.

Limiti dell'astrazione funzionale

1. I dettagli relativi alla rappresentazione dei dati di input e output devono essere conosciuti da chi poi andrà a realizzare il modulo.

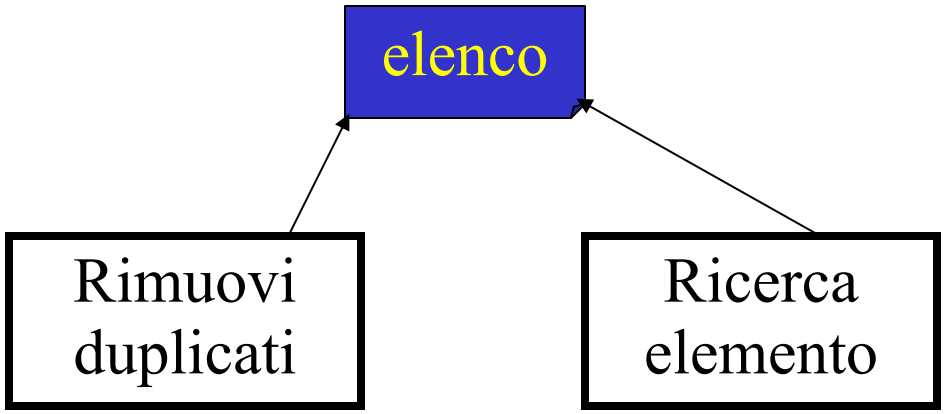
Esempio: un modulo che rimuove i duplicati in un elenco deve sapere se questo è realizzato con un array, un file, etc.

2. La rappresentazione è solitamente condivisa fra diversi moduli, per cui i cambiamenti apportati alla rappresentazione dei dati in input/output a un modulo si possono ripercuotere su molti altri moduli.

Limiti dell'astrazione funzionale

Esempio: due moduli operano su uno stesso elenco, uno per rimuovere duplicati e l'altro per ricercare un elemento.

- Requisito: migliorare l'efficienza del modulo di ricerca
- Possibile soluzione: ricorso ad una rappresentazione dell'elenco con tabelle hash.
- Impatto della soluzione adottata: un cambiamento nel primo modulo, che dovrà necessariamente operare la trasformazione (rimozione dei duplicati) in modo differente.



Limiti dell'astrazione funzionale

- non permette di sviluppare soluzioni invarianti ai cambiamenti nei dati (sono invarianti solo ai cambiamenti nei processi di trasformazione che operano).
  - rende difficoltosa la manutenzione delle soluzioni progettate.
- 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*

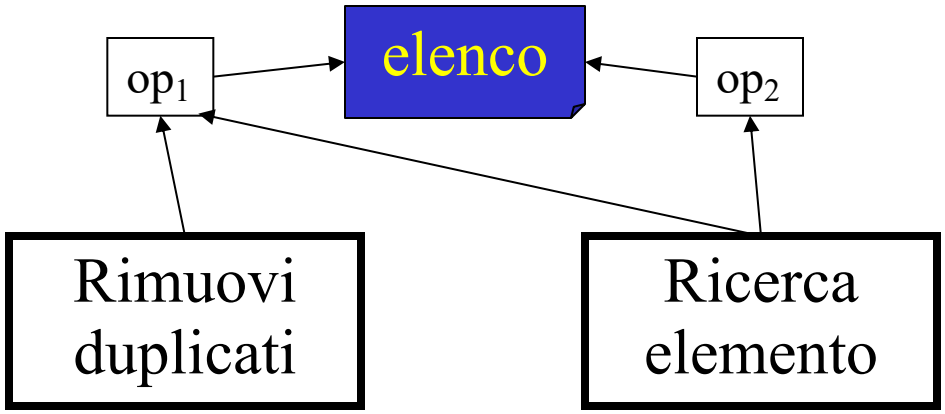
  (principio dell'astrazione dati).

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’ accedessero all’elenco attraverso un insieme di operazioni lecite (e.g., ‘dammi il prossimo elemento’, ‘non ci sono più elementi’, ecc.) il cambiamento della rappresentazione dell’elenco richiederebbe 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

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

Esempi:

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

Incapsulamento

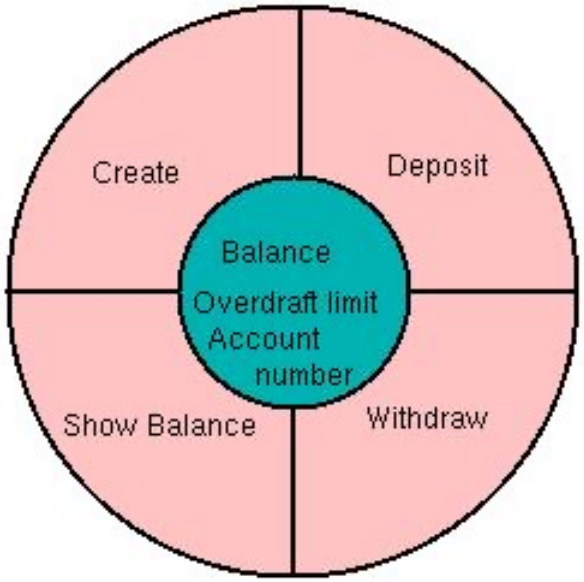
- L'incapsulamento NON dice come devono essere le "pareti" del pacchetto (o capsula), che potranno essere:
  - Trasparenti: permettendo di vedere tutto ciò che è stato impacchettato;
  - Traslucide: permettendo di vedere in modo parziale il contenuto;
  - Opache: nascondendo tutto il contenuto del pacchetto.

Astrazione dati & Incapsulamento: combinazione

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



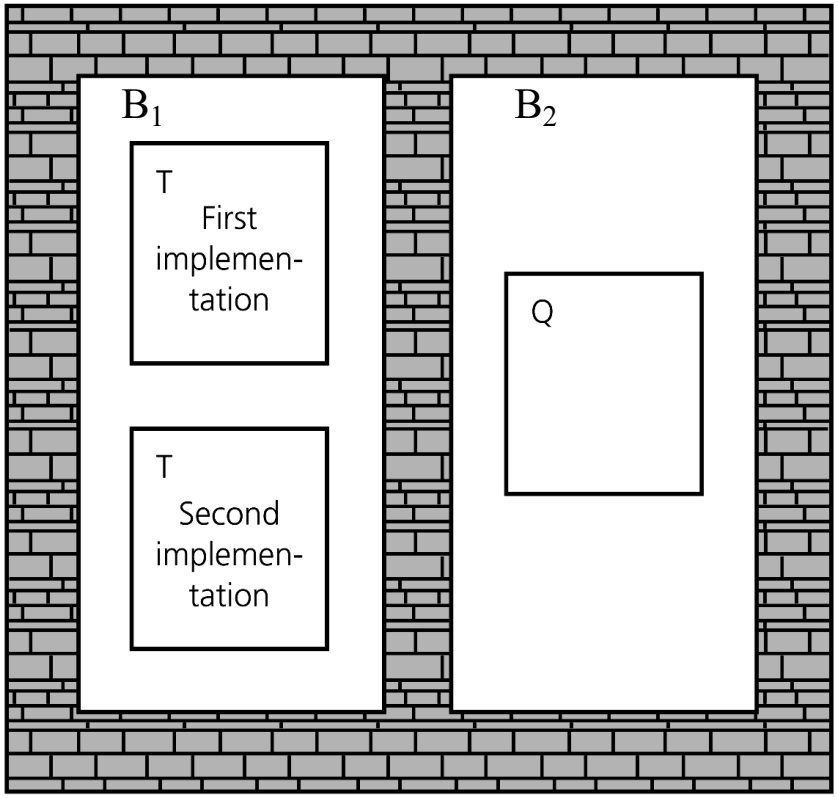
- Esempio: il dato "conto corrente" ha una sua rappresentazione interna che permette di memorizzare
  - il saldo (*balance*),
  - il limite fido (*overcraft limit*),
  - il numero di conto (*account number*).

  La rappresentazione interna dei tre dati è nascosta. L'accesso alla rappresentazione passa per 4 operazioni lecite:
  - Creazione conto;
  - Deposito;
  - Prelievo:
  - Stampa saldo.

Rappresentazione ed operazioni lecite sono impacchettate in un modulo

Astrazione dati & Incapsulamento

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



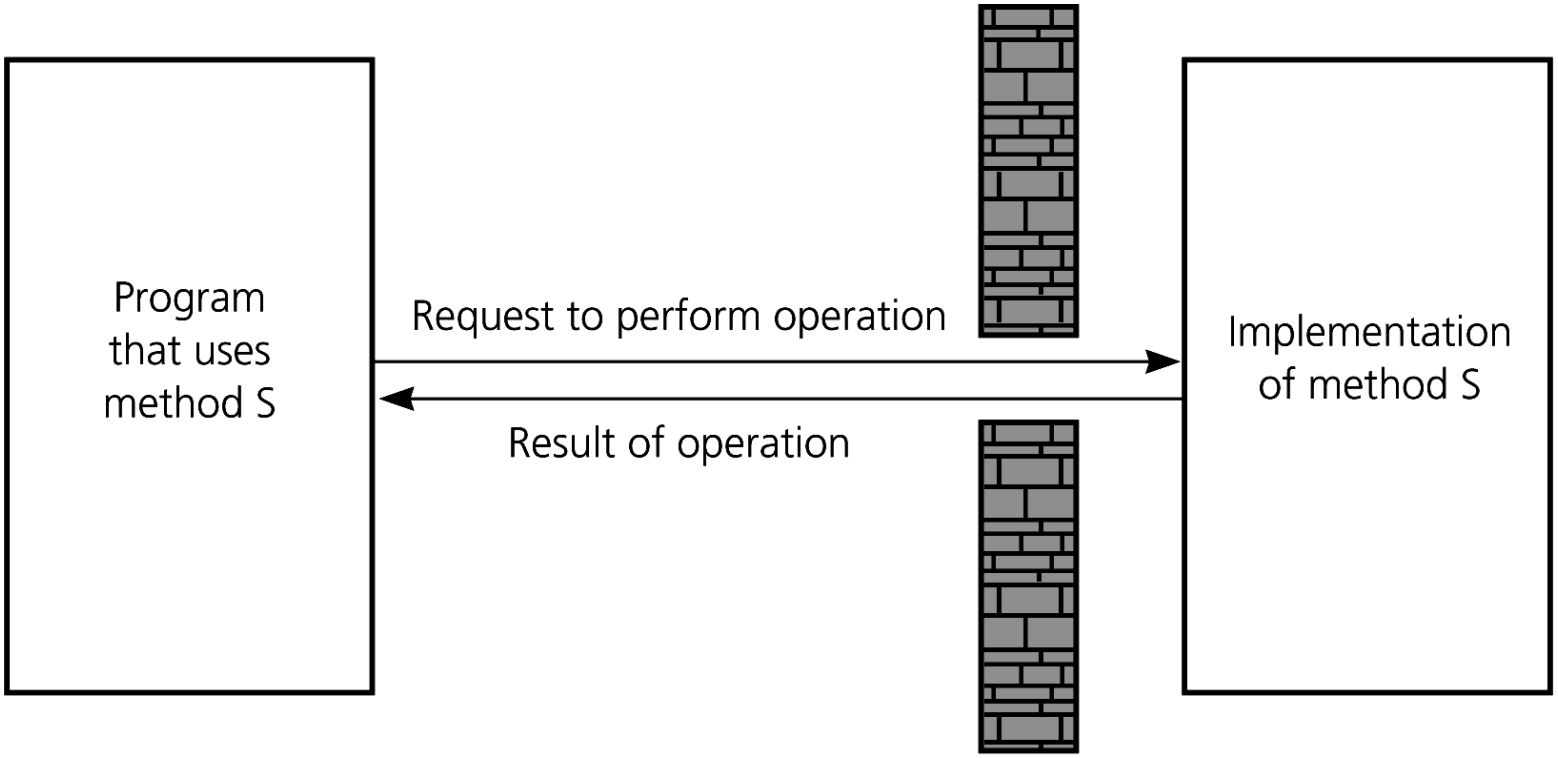
Astrazione dati vs. Astrazione funzionale

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

- L'astrazione dati ricalca ed estende quella funzionale.
  - L'esperienza ha 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.

Astrazione dati & Incapsulamento: limiti

- 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 potrebbe essere

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

(si riveda l'esempio del fattoriale)

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
- I meccanismi utilizzati per realizzare queste associazioni sono fondamentalmente due:
  - **Meccanismi di copia**: copiano il valore da passare
  - **Meccanismi definizionali:** legano direttamente il parametro formale alla definizione dell'argomento passato

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

Astrazione dati

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

Uso dell'astrazione dati

- 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: linguaggi di specifica

- I linguaggi di specifica per astrazione dati più noti sono due:
  - logico-matematico usato nelle asserzioni ➔ specifiche assiomatiche
  - Algebrico 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:

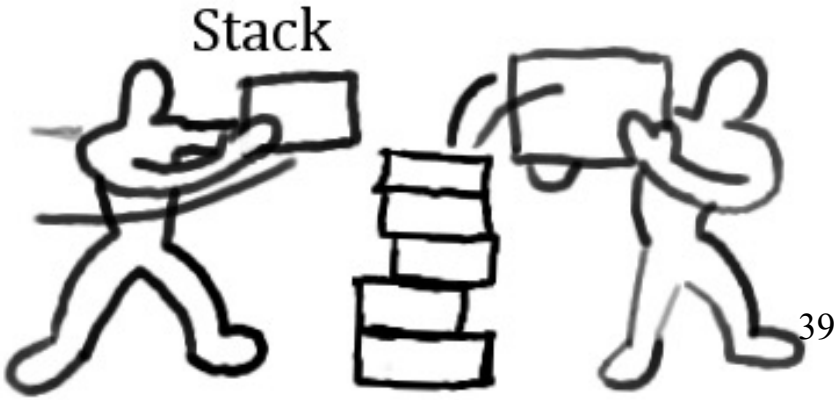
i) *precondizione*, che definisce quando l'operatore è applicabile

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

Il dato astratto *Pila*

Una pila (*stack*) è una sequenza di elementi di un certo tipo in cui è possibile aggiungere o togliere elementi solo da un estremo della sequenza (la "testa").

– Può essere intesa come un caso speciale di lista in cui l'ultimo elemento inserito è il primo ad essere rimosso (LIFO, ***last in, first out***) e non è possibile accedere ad alcun elemento che non sia quello in testa.



Il dato astratto *Pila*

Una specifica assiomatica di una pila

Specifica sintattica

Tipi: pila, booleano, tipoelem

Operatori:

CREAPILA() → pila

PILAVUOTA(pila) → booleano

LEGGIPILA(pila) → tipoelem

FUORIPILA(pila) → pila

INPILA(tipoelem, pila) → pila

Il tipo astratto *Pila*

Specifica semantica

Tipi:

pila = insieme delle sequenze di elementi di tipo tipoelem

booleano = insieme dei valori di verità {vero,falso}

Operatori:

CREAPILA ()=P'

Post: P' = ^ , la sequenza vuota

PILAVUOTA(P)=b

Post: b=vero, se P=^; b=falso, altrimenti

Il tipo astratto *Pila*

LEGGIPILA(P)=a

Pre: $P = a_1, a_2, \ldots, a_n$ e $n \geq 1$

Post: $a = a_1$

FUORIPILA(P)=P'

Pre: $P = a_1, a_2, \ldots, a_n$ e $n \geq 1$

Post: $P' = a_2, \ldots, a_n$ se $n>1$; P' =^ se $n=1$

INPILA(a,P)=P'

Pre: $P = a_1, a_2, \ldots, a_n$ e $n \geq 0$

Post: $P' = a, a_1, a_2, \ldots, a_n$ se n>0; P' = a se n=0

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'***	
	creapila	*inpila(stk, i)*
fuoripila(stk')	error	stk
leggipila(stk')	error	i
pilavuota(stk')	true	false

Le specifiche di seguito mostrate sono algebriche, in particolare quanto mostrato afferisce alla parte di restrizione di tale specifiche

Parte di restrizione: stabilisce varie condizioni che devono essere soddisfatte o prima che siano applicate le operazioni o dopo che esse siano state completate.

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***	
	creapila	*inpila(stk, i)*
creapila	true	false
inpila(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.

La scelta dei costruttori: esempio

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, in quanto ottenibile a partire da add(string,char) per cui non è scelto come costruttore.

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

Astrazione di controllo

- L'astrazione di controllo si riferisce alla possibilità di specificare un modulo software che **esegue** delle operazioni in un **ordine**, nascondendo i dettagli su come il mantenimento dell'ordine è ottenuto.
- In sintesi:
  - Il modulo software deve essere parametrizzato rispetto alle operazioni da eseguire;
  - Il modulo software è associato a un controllo di sequenza (*control flow*)
  - I dettagli di come il controllo di sequenza è garantito non sono visibili al consumatore (fruitore) del modulo.

Astrazione di controllo: esempio

Un dato astratto che contiene un insieme di valori può mettere a disposizione due operatori:

- *next* che restituisce un valore
- *hasnext* che è vero se ci sono ancora altri valori da selezionare.

L'astrazione di controllo *for-each(dato astratto, istruzione)* utilizzerà questi due operatori per eseguire l'*istruzione* su ogni valore contenuto nel dato astratto. L'informazione sull'ordine con cui i valori sono considerati (cioè sul protocollo di iterazione) è nascosta all'utente del *for-each*.

- andando ancora oltre, si può ipotizzare che il dato astratto deleghi un modulo **iteratore** a fornire i servizi richiesti dal *for-each*. In questo modo i meccanismi di visita sono tutti incapsulati in un modulo.
- la tecnica dell'incapsulamento viene in aiuto al progettista che vuole applicare una qualche forma di astrazione.

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

Approfondimenti ed esercizi

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: $0 \le i \le n-1$

Post: e=v(i)

SCRIVIVETTORE(v,i,e) = v'

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

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

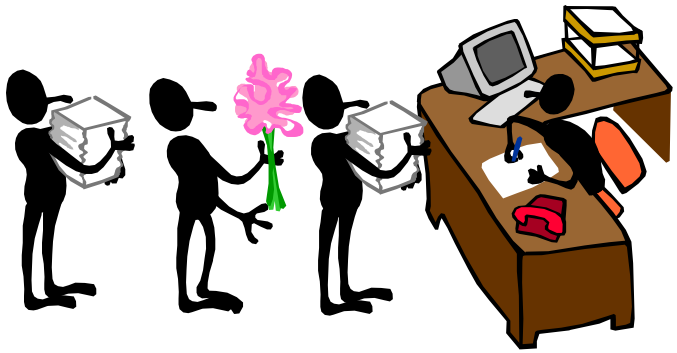
In Java il vettore (array) è un tipo di dato concreto. La corrispondenza tra la specifica appena introdotta e quella del linguaggio Java è la seguente:

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.

Il dato astratto *Coda*

- Una coda è un dato astratto che consente di rappresentare una sequenza di elementi in cui è possibile aggiungere elementi ad un estremo (“il fondo”) e togliere elementi dall’altro estremo (“la testa”).
  - Tale disciplina di accesso è detta FIFO (First In First Out).
  - È adatta a rappresentare sequenze nelle quali l’elemento viene elaborato secondo l’ordine di arrivo (lista d’attesa, etc.)



Il dato astratto *Coda*

Una specifica assiomatica di una coda

Specifica sintattica

Tipi: coda, booleano, tipoelem

Operatori:

CREACODA() → coda

CODAVUOTA(coda) → booleano

LEGGICODA(coda) → tipoelem

FUORICODA(coda) → coda

INCODA(tipoelem, coda) → coda

Il dato astratto *Coda*

Specifica semantica

Tipi:

coda = insieme delle sequenze $\langle a_1, a_2, \ldots, a_n \rangle$, con n>0, di elementi di tipo *tipoelem*. Λ denota la sequenza vuota.

booleano = insieme dei valori di verità {vero,falso}

Operatori:

CREACODA()=q'

Post: q'= Λ

CODAVUOTA(q)=b

Post: b=vero, se q=Λ; b=falso, altrimenti

LEGGICODA(q)=a

Pre: $q = \langle a_1, a_2, \ldots, a_n \rangle \neq \Lambda$

Post: $a = a_1$

FUORICODA(q)=q'

Pre: $q = \langle a_1, a_2, \ldots, a_n \rangle \neq \Lambda$

Post: $q' = \langle a_2, \ldots, a_n \rangle$ se n>1 ; $q' = \Lambda$ *se n=1*

Il dato astratto *Coda*

Specifica semantica (cont.)

INCODA(a,Q)=Q'

Pre: $q = \langle a_1, a_2, \ldots, a_n \rangle$ oppure $q = \Lambda$

Post: $q' = \langle a_1, a_2, \ldots, a_n, a \rangle$, se $q \neq \Lambda$;

$q' = \langle a \rangle$ se $q = \Lambda$

Esercizio Deque (*homework*)

- Si progetti un dato astratto *Deque* (*double-ended queue* o *coda doppia*), una struttura lineare che permette di inserire, cancellare, esaminare elementi solo dalle due estremità.
- Dare le specifiche assiomatiche (semantiche) di *deque*, tenuto conto delle seguenti specifiche sintattiche (*vedere slide successive*)

Esercizio *Deque*

Tipi

deque, item, boolean

Operatori

new() → deque	crea nuove code doppie
addq(item, deque) → deque	accoda un elemento
bottom(deque) → item	restituisce l'elemento di coda
leaveq(deque) → deque	estrae l'elemento di coda
push(item, deque) → deque	impila un elemento
top(deque) → item	restituisce l'elemento di testa
pop(deque) → deque	estrae l'elemento di testa
isnew(deque) → boolean	predica se una coda doppia è vuota

Esercizio Deque (soluzione) …

Specifica semantica

Tipi:

deque = insieme delle sequenze $\langle a_1, a_2, \ldots, a_n \rangle$, con n>0, di elementi di tipo *item*. Λ denota la sequenza vuota.

boolean = insieme dei valori di verità {vero, falso}

Operatori:

new ()=q'

Post: $q' = \Lambda$

… Esercizio Deque (soluzione) …

Specifica semantica (cont.)

addq(a,q)=q' // accoda un elemento

Pre: $q = \langle a_1, a_2, \ldots, a_n \rangle$ oppure $q = \Lambda$

Post: $q' = \langle a_1, a_2, \ldots, a_n, a \rangle$, se $q \neq \Lambda$;

$q' = \langle a \rangle$ se $q = \Lambda$

bottom(q)=a // restituisce l'elemento di coda

Pre: $q = \langle a_1, a_2, \ldots, a_n \rangle \neq \Lambda$

Post: $a = a_n$

leaveq(q)=q' //estrae l'elemento di coda

Pre: $q = \langle a_1, a_2, \ldots, a_n \rangle \neq \Lambda$

Post: $q = \langle a_1, a_2, \ldots, a_{n-1} \rangle$ se n>1 ; $q' = \Lambda$ *se* $n=1$

… Esercizio Deque (soluzione) …

Specifica semantica (cont.)

push(a,q)=q' // impila un elemento

Pre: $q = \langle a_1, a_2, \ldots, a_n \rangle$ oppure $q = \Lambda$

Post: $q' = \langle a, a_1, a_2, \ldots, a_n \rangle$, se $q \neq \Lambda$;

$q' = \langle a \rangle$ se $q = \Lambda$

top(q)=a // restituisce l'elemento di testa

Pre: $q = \langle a_1, a_2, \ldots, a_n \rangle \neq \Lambda$

Post: $a = a_1$

pop(q)=q' // estrae l'elemento di testa

Pre: $q = \langle a_1, a_2, \ldots, a_n \rangle \neq \Lambda$

Post: $q = \langle a_2, \ldots, a_n \rangle$ se $n > 1$; $q' = \Lambda$ se $n = 1$

… Esercizio Deque (soluzione)

Specifica semantica (cont.)

isnew(q)=b // predica se una coda doppia è vuota

Post: b=vero, se q=Λ; b=falso, altrimenti

Esercizio (*homework*)

Si progetti il dato astratto *volo*. Si forniscano le specifiche semantiche di volo, le cui specifiche sintattiche sono:

Tipi: volo, passeggero, posto, integer, boolean

Operatori

- creavolo() → volo crea un nuovo volo
- postolibero(posto, volo) → boolean predica se il posto è libero o meno
- check-in(posto, passeggero, volo) → volo aggiunge un passeggero al volo assegnandogli il posto specificato, purché libero
- libera(posto, volo) → volo libera un posto sul volo
- sostituisci(posto, passeggero,volo) → volo assegna a un altro passeggero un posto già occupato
- passeggero(posto, volo) → passeggero restituisce il passeggero che occupa un posto
- passeggeri(volo) → integer restituisce il numero di passeggeri con carta di imbarco, cioè che hanno fatto il check-in

Esercizio volo: suggerimenti

posto è l'insieme K dei posti

passeggero è l'insieme P dei passeggeri.

volo è l'insieme di voli, dove ogni volo è a sua volta un insieme di coppie (k, p) $\in K \times P$.

integer è l'insieme Z dei numeri interi. 0 e 1 sono particolari valori dell'insieme Z.

boolean è l'insieme dei valori di verità {true,false}.

N.B.: Questa definizione è sufficientemente astratta e non dice nulla sull'insieme K. Infatti i posti su un aereo sono generalmente identificati da una coppia (*Numero Lettera, Intero)*

Esercizio volo (soluzione) …

Specifica semantica

Tipi:

volo = insieme di insiemi di coppie (k,p) ∈ K×P dove K è l'insieme dei posti e P è l'insieme dei passeggeri. ∅ denota l'insieme vuoto.

integer = insieme di numeri naturali

boolean = insieme dei valori di verità {vero, falso}

Operatori:

creavolo() → v' // crea un nuovo volo

Post: v′ = ∅

… Esercizio volo (soluzione) …

Specifica semantica (cont.)

postolibero(k, v) =b //predica se il posto è libero o meno

Pre: $v \subseteq K \times P$, $k \in K$

Post: b=vero se $\neg(\exists\, p \in P\ (k,p) \in v)$, falso altrimenti

check-in(k, p, v) = v' // aggiunge un passeggero al volo assegnandogli il posto specificato, purché libero

Pre: $v \subseteq K \times P$, $k \in K$, $p \in P$ $\neg(\exists\, q \in P\ (k, q) \in v)$,

Post: $v' = v \cup \{(k,p)\}$

… Esercizio volo (soluzione) …

Specifica semantica (cont.)

libera(k, v)= v' //libera un posto sul volo

Pre: $v \subseteq K \times P$, $k \in K$, $\exists\, q \in P\ (k, q) \in v$

Post: $v' = v - \{(k,q)\}$

sostituisci(k, p, v)=v' //assegna a un altro passeggero un posto già occupato

Pre: $v \subseteq K \times P$, $k \in K$, $p \in P$, $\exists q \in P\ (k,q) \in v$

Post: $v' = (v - \{(k,q\}) \cup \{(k,p)\}$

… Esercizio volo (soluzione) …

Specifica semantica (cont.)

passeggero(k, v)=p //restituisce il passeggero che occupa un posto

Pre: $v \subseteq K \times P$, $k \in K$, $\exists q \in P\ (k, q) \in v$

Post: $p=q$

passeggeri(v)=n //restituisce il numero di passeggeri con carta di imbarco, cioè che hanno fatto il check-in

Pre: $v \subseteq K \times P$

Post: $n=|v|$