2. Algebre di Dati

Dati e rappresentazioni, requisiti delle astrazioni di dati, costrutti. Astrazioni di dati e dati primitivi. Specifica sintattica e semantica. La realizzazione.

Algoritmi e Strutture Dati + Lab
A.A. 18/19

Informatica
Università degli Studi di Bari “Aldo Moro”

Nicola Di Mauro

Astrazione funzionale

- Un programma corrisponde alla tripla {D, A, R}
  - Si può dire che, poiché trasforma dati iniziali in risultati, definisce un nuovo operatore sui dati
  - Il repertorio di operatori disponibili sui dati può essere ampliato scrivendo programmi
- L'astrazione funzionale è la tecnica che permette di potenziare il linguaggio disponibile introducendo nuovi operatori
  - Questi sono definiti attraverso sotto-programmi in cui si fa uso degli operatori di base già disponibili
  - L'intestazione del sotto-programma introduce il nome e gli argomenti dell'operatore, mentre il corpo ne descrive le regole di comportamento

Astrazione funzionale

- Consideriamo la formula per il calcolo del coefficiente binomiale

$$\frac{n!}{k!(n-k)!} = \binom{n}{k}$$

  che fornisce il numero di possibili combinazioni di *n* oggetti presi a gruppi di *k*

  - Il calcolo è immediato se si suppone di avere un operatore Fatt(*i*) che calcola il fattoriale per un generico valore di *i*

```
int Binomial(int n, int k)
    return Fatt(n)/(Fatt(k) * Fatt(n-k));
```

  - Se Fatt(*i*) non è disponibile è necessario scrivere un sotto-programma per ampliare il repertorio degli operatori
  - L'astrazione funzionale ci consente questo

Astrazione funzionale

- I linguaggi di programmazione ad alto livello permettono l'uso di astrazione funzionale, cioè consentono di
  - creare delle unità di programma dando un nome ad un gruppo di istruzioni e stabiliscono le modalità di comunicazione tra l'unità di programma creata ed il resto del programma in cui essa si inserisce.
  - al momento della attivazione (su chiamata) della unità di programma viene sospesa l'esecuzione del programma (o dell'unità) chiamante: il controllo passa alla unità attivata e, all'atto del completamento della sua esecuzione, l'attivazione termina e il controllo torna al programma chiamante.

Astrazione funzionale

- I costrutti linguistici per realizzare la astrazione funzionale consentono di definire una specifica e una realizzazione
  - La specifica definisce “cosa” ci si aspetta dalla funzione, cioè definirà
    - “la relazione che caratterizza il legame tra dati di ingresso e risultati”
  - Nel caso del fattoriale la specifica dirà che:
    - L'intestazione è int Fatt(int k); l'ingresso k (argomento) è una variabile di tipo intero; il risultato è ancora di tipo intero
    - La funzione da calcolare è definita come
      Fatt(k)=k (k-1) … (2) (1)

Astrazione funzionale /2

- La realizzazione definisce “come” il risultato viene ottenuto
  - Alla stessa specifica, infatti, potranno corrispondere scelte di realizzazione diverse
  - La funzione Fatt potrebbe essere realizzata con un programma iterativo

```
Fatt(int k)
    f = 1
    for i = 2 to k do
        f = f * i
    return f
```

  - ma anche con un programma ricorsivo

```
Fatt(int k)
    if k = 1 then f = 1
    else f =  k * Fatt(k-1)
    return f
```

Astrazione dati

- La astrazione di dati ricalca ed estende il concetto di astrazione funzionale
- Così come l'astrazione funzionale permette di ampliare l'insieme dei modi di operare sui dati, cioè gli operatori sui tipi di dati già disponibili, la astrazione di dati permette di ampliare i tipi di dati disponibili attraverso l'introduzione sia di nuovi tipi di dati che di nuovi operatori
- L'astrazione funzionale stimola gli sforzi per evidenziare operazioni ricorrenti o ben caratterizzate all'interno della soluzione di un problema
- L'astrazione di dati sollecita ad individuare le organizzazioni dei dati più adatte alla soluzione del problema

Astrazione dati

- Un'astrazione di dati consente una estensione dell'algebra dei dati disponibile in un linguaggio di programmazione
- Cosa è un'algebra? è un sistema matematico costituito da un dominio, cioè un insieme di valori, e da un insieme di funzioni applicabili su tali valori
  - il numero, il tipo e le proprietà delle funzioni sono gli elementi essenziali di un algebra
- La corrispondenza tra tipo astratto e algebra si basa sul parallelo tra gli insiemi di valori di un algebra e i domini di definizione di un tipo astratto, e tra le funzioni di un'algebra e le operazioni associate ad un tipo astratto.

Astrazione dati /2

- Ma come estendere l'algebra dei dati disponibile in un linguaggio di programmazione?
- Molti linguaggi di programmazione forniscono i mezzi per definire e creare nuovi tipi di dati, ma non tutti consentono di fare astrazione dati o creare tipi di dati astratti.
- Un tipo di dati si dice astratto se le operazioni applicabili sugli oggetti che li rappresentano sono isolate dai dettagli usati per realizzare il tipo

Astrazione dati /3

- Un'algebra dei dati può essere definita come costituita da:
  - A) una famiglia di insiemi (insieme di dati)
  - B) una famiglia di operatori sui dati (operatori)
  - C) un repertorio di simboli (o nomi) per indicare l'insieme di dati
  - D) un repertorio di simboli (o nomi) per indicare gli operatori
  - E) un repertorio di simboli (detti costanti) per indicare elementi singoli degli insiemi di dati

Algebra di dati: un esempio

- A) La famiglia di insiemi comprende:
  - i numeri interi, i booleani, le stringhe (sequenze di caratteri alfanumerici)
- B) La famiglia di operatori comprende:
  - operatori aritmetici (somma, sottrazione, moltiplicazione e divisione)
  - operatori di congiunzione, disgiunzione e negazione per i booleani
  - operatori di confronto tra interi (maggiore, minore, uguale etc.)
  - operatori di concatenazione per le sequenze di caratteri e di selezione di sottostringa (questo restituisce come risultato la parte di stringa compresa tra due numeri interi che indicano la posizione del primo e dell'ultimo carattere)

Algebra di dati: un esempio /2

- C) Il repertorio dei simboli o nomi per l'insieme di dati potrebbe essere:
  - integer, boolean, string
- D) Il repertorio di simboli o nomi per gli operatori potrebbe essere:
  - + , - , * , / per gli operatori aritmetici
  - and , or , not per gli operatori booleani
  - > , < , = per i confronti tra interi
  - substr e & stringhe per la selezione e la concatenazione di stringhe

Algebra di dati: un esempio /3

- E) Gli elementi singoli per gli insiemi di dati (le costanti) potrebbero essere:
  - ogni intero viene indicato da una sequenza di cifre decimali, eventualmente preceduta da + o - , interpretata secondo la notazione decimale;
  - ogni sequenza di caratteri viene indicata dalla sequenza racchiusa tra apici;
  - i valori booleani sono indicati da true e false.

Gli operatori

- Gli operatori possono essere combinati per ottenere delle espressioni
- Le caratteristiche di un'algebra dipendono dalla famiglia di insiemi e dalla famiglia di operatori
- Nei linguaggi di programmazione il binomio “dati–operatori” è inscindibile
- Dunque, è necessario definire un tipo di dati come un insieme di dati più gli operatori che ad esso si riferiscono, nel contesto di una definita algebra

Astrazione dati

- Anche un'astrazione di dati è costituita da:
  - una specifica
    - ha il compito di descrivere sinteticamente il tipo dei dati e gli operatori che li caratterizzano
  - una realizzazione
    - stabilisce come i dati e gli operatori vengono ricondotti ai tipi di dati e agli operatori già disponibili
- L'astrazione di dati stimola gli sforzi per individuare le organizzazioni di dati più adatte alla soluzione di un problema mentre la astrazione funzionale porta a evidenziare operazioni ricorrenti o funzionalità caratterizzate all'interno di un problema.
- La scelta delle strutture dati è il primo passo per un buon risultato nell'attività di programmazione

Obiettivo

- Attuare una corretta astrazione di dati
  - bisogna identificare i costrutti di programmazione che la consentono
  - non è sufficiente usare sottoprogrammi che implementano i nuovi operatori

Astrazione dati: un esempio

- Supponiamo di voler effettuare un'astrazione di dati aggiungendo ai tipi di dati standard i numeri complessi
- Come è noto i numeri complessi hanno forma: r1 + i r2
  - Dove r1 e r2 sono numeri reali e i è √(-1)
- Per denotarli matematicamente basta la coppia (r1, r2)
  - dove r1 è il coefficiente della parte reale e r2 quello della parte immaginaria
- E' necessario definire operatori di selezione e costruzione che operano su complessi e su reali, operatori di somma e prodotto che operano esclusivamente sui complessi

Astrazione dati: un esempio /2

- Il calcolo di somma e prodotto su numeri complessi non è difficile se ricordiamo che
  - la somma di due numeri complessi c1 e c2 dà luogo ad un nuovo numero complesso c3 la cui parte reale è la somma delle parti reali e la cui parte immaginaria è la somma delle parti immaginarie
    - parteReale(c3) = parteReale(c1) + parteReale(c2)
    - parteImmg(c3) = parteImmg(c1) + parteImmg(c2)

Astrazione dati: un esempio /2

- Il prodotto di due numeri complessi c1 e c2 dà luogo ad un nuovo numero complesso c3
  - la cui parte reale è la differenza tra il prodotto della parte reale del primo numero per la parte reale del secondo numero e il prodotto della parte immaginaria del primo numero per la parte immaginaria del secondo numero
    - parteReale(c3) = parteReale(c1) * parteReale(c2) - parteImmg(c1) * parteImmg(c2)
  - e la cui parte immaginaria è la somma del prodotto della parte reale del primo numero per la parte immaginaria del secondo numero con il prodotto della parte immaginaria del primo numero con la parte reale del secondo numero
    - parteImmg(c3) = parteReale(c1) * parteImmg(c2) + parteImmg(c1) * parteReale(c2)

Astrazione dati: un esempio /3

- Passando agli operatori necessari, abbiamo bisogno di definire, attraverso procedure e funzioni:
  - cxsum(c1, c2) = c3 — somma di complessi
  - cxprod(c1, c2) = c3 — prodotto di complessi
  - realpart(c) = r — seleziona parte reale del complesso c
  - immgpart(c) = i — seleziona parte immaginaria del complesso c
  - conscx(r1, r2) = c — combinazione di due reali, il primo come parte reale, il secondo come immaginaria del complesso c

Astrazione dati: un esempio /4

- Per poter fare astrazione dati, basta disporre di linguaggi che consentono astrazione funzionale?
  - Con il fortran, ad esempio, si può fare astrazione funzionale con l'uso di subroutine, ma non è possibile fare astrazione dati
  - I numeri complessi sono rappresentati come coppie (r1, r2)
    - i vettori possono essere usati a tale scopo
      - dimension c1(2), c2(2), c3(2), c4(2) è la dichiarativa fortran per i vettori che rappresentano i numeri complessi
    - La somma può essere realizzata con un sottoprogramma
      - subroutine sommacomplessi (c1(2), c2(2), c3(2))
      - La stessa cosa può farsi per gli altri operatori.
      - Va, però, evidenziato che input a questa subroutine possono essere passati tutti i vettori di interi a due elementi, anche quelli che non hanno nulla a che vedere con la nostra astrazione
    - Non vi è nessun modo, in questo tipo di linguaggio, di consentire che solo i dati astratti definiti abbiano l'accesso agli operatori appositamente costruiti (nel nostro caso ai vettori che rappresentano i complessi)
  - Il discorso è diverso se ho a disposizione un linguaggio a tipizzazione forte come il pascal

Astrazione dati: un esempio /5

- In pascal si può dichiarare
  - TYPE
    COMPLEX = ARRAY [2] OF REAL;
    VAR
    C1, C2, C3, C4 : COMPLEX;

    PROCEDURE SOMMACOMPLESSI (
    C1, C2 : COMPLEX;
    VAR C3 :COMPLEX);
- Oppure
  - TYPE
    COMPLEX= RECORD OF
    PARTEREALE : REAL;
    PARTEIMMAGINARIA :REAL;
    END;

Astrazione dati

- Disponendo di un linguaggio con tipi, possiamo:
  - utilizzare dati che definiamo e dichiariamo direttamente come complessi, prescindendo dalla effettiva realizzazione
  - fare in modo che alle procedure (operatori) costruite per i nostri dati abbiano accesso esclusivamente quei dati
- Queste caratteristiche necessarie a una buona astrazione di dati sono note come
  - i requisiti della astrazione di dati

Requisiti della astrazione di dati

- **Requisito di astrazione**
  - è verificato quando i programmi che usano un'astrazione possono essere scritti in modo da non dipendere dalle scelte di realizzazione
  - Nel precedente esempio, in pascal le diverse realizzazioni non provocano cambi nelle intestazioni dei sottoprogrammi che realizzano gli operatori
- La mancanza del requisito di astrazione
  - si manifesta con l'impossibilità di dichiarare “direttamente” variabili con il nuovo tipo definito
    - non posso, nel codice, fare riferimento ai nuovi dati creati prescindendo dalla rappresentazione usata e dalla realizzazione

Requisiti della astrazione di dati

- **Requisito di protezione**
  - è verificato se sui nuovi dati si può lavorare esclusivamente con gli operatori definiti all'atto della specifica
- La mancanza del requisito di protezione
  - si manifesta con la possibilità di lavorare con gli operatori definiti per i nuovi dati anche su dati con rappresentazioni simili, ma non omogenei per tipo
  - nell’esempio appena visto, relativo ai numeri complessi, in fortran, il tipo di dichiarative e la realizzazione con vettori e subroutine consente l'accesso agli operatori da parte di qualunque vettore di reali

Requisiti della astrazione di dati

- Il requisito di astrazione, in accordo con i principi della astrazione di dati, impone che i programmi siano sensibili solo a variazioni della specifica dei nuovi tipi
- Il requisito di protezione è particolarmente importante, se vogliamo che l'esecutore del linguaggio assicuri i controlli di consistenza tra i tipi di dati e gli operatori, non solo ai dati primitivi, ma anche ai dati ottenuti per astrazione

Specifica e Realizzazione

- Si è detto che un'astrazione di dati è costituita da:
  - una **specifica** e una **realizzazione**
- Una specifica, che ha il compito di descrivere sinteticamente il tipo dei dati e gli operatori che li caratterizzano, si distingue in:
  - a) **specifica sintattica** che fornisce:
    - l'elenco dei nomi dei tipi di dato utilizzati per definire la struttura, delle operazioni specifiche della struttura e delle costanti
    - i domini di partenza e di arrivo, cioè i tipi degli operandi e del risultato per ogni nome di operatore

Specifica e Realizzazione /2

- b) **specifica semantica** che definisce il significato dei nomi introdotti con la specifica sintattica. Associa:
  - un insieme ad ogni nome di tipo introdotto nella specifica sintattica
  - un valore ad ogni costante
  - una funzione ad ogni nome di operatore esplicitando le seguenti condizioni sui domini di partenza e di arrivo:
    - **precondizione** che definisce quando l'operatore è applicabile
    - **postcondizione** che stabilisce come il risultato sia vincolato agli argomenti dell'operatore

Specifica e Realizzazione /3

- La **realizzazione** sfrutta al meglio le possibilità offerte dall'ambiente di programmazione
- La definizione di un formalismo per le specifiche sintattiche non pone particolari problemi
- L'individuazione di formalismi per le specifiche semantiche è assai più difficile
  - in genere, le specifiche semantiche sono date con frasi estratte dal linguaggio naturale o da quello matematico

Specifiche: un esempio

- I numeri interi e i booleani
  - Specifica sintattica

    Tipi:
    integer, boolean
    Operatori:
    + , - : (integer, integer) → integer
    < , > : (integer, integer) → boolean
    and, or : (boolean, boolean) → boolean
    not : (boolean) → boolean
    seqcif : ( ) → integer
    true : ( ) → boolean
    false : ( ) → boolean

Specifiche: un esempio /2

- I numeri interi e i booleani
  - Specifica semantica

    Tipi :
    integer: l'insieme dei numeri interi
    boolean: l'insieme dei valori di verità

    Operatori:
    + e - : forniscono come risultato somme e differenze
    < e > : danno vero se il primo operatore è minore (maggiore) del secondo
    and e or: forniscono la congiunzione e la disgiunzione logiche dei due operandi

Specifiche: un esempio /3

- Operatori (cont.)

  seqcif : fornisce l'intero secondo notazione decimale
  true: corrisponde al valore di verità vero
  false: corrisponde al valore di verità falso

- In questa specifica il tipo di dato integer possiede un numero infinito di costanti (ogni possibile sequenza di cifre decimali è una costante)
- Ma vi è un solo modo di definire una specifica?
- L'insieme degli operatori definibili su insiemi di dati è unico?

Specifiche: un esempio /4

- Avremmo potuto utilizzare un'altra specifica
  - Specifica sintattica

    Tipi:
    integer, boolean

    Operatori:

zero	: ( )	→ integer
true	: ( )	→ boolean
false	: ( )	→ boolean
succ	: (integer)	→ integer
pred	: (integer)	→ integer
iszero	: (integer)	→ boolean

Specifiche: un esempio /5

- Avremmo potuto utilizzare un'altra specifica
  - Specifica semantica

    Tipi :
    integer : l'insiemi dei numeri interi
    boolean : l'insieme dei valori di verità

    Operatori:
    zero : dà lo zero dei numeri interi
    true : dà il valore di verità vero
    false : dà il valore di verità falso
    succ (n) : dà n+1
    pred (n) : dà n – 1
    iszero(n) : dà vero se e solo se n è lo zero degli interi

Realizzazione

- La realizzazione riconduce la specifica ai tipi di dato primitivi e agli operatori già disponibili
  - in particolare gli operatori sono simulati con sottoprogrammi
- Alla stessa specifica possono corrispondere diverse realizzazioni più o meno efficienti
  - le realizzazioni possono essere trasparenti all'utente
- Nel valutare l'efficienza di sottoprogrammi che utilizzano tipi di dato primitivi, si prescinde dalle caratteristiche della macchina, assumendo di far riferimento ad una organizzazione molto semplice

Realizzazione

- Si fa riferimento a
  - dati contenuti in memoria
  - memoria organizzata in celle (parola)
  - celle accessibili tramite indirizzo
  - indirizzi che sono interi consecutivi
  - dati elementari che non decidono la capacità di una cella (maxint)
  - variabili contenute in celle di indirizzo noto il cui accesso richiede tempo costante

Realizzazione /2

- Un costrutto di programmazione propriamente adatto alla realizzazione di astrazione di dati dovrà:
  - dichiarare esplicitamente quali sono i nuovi tipi di dati e quali sono i nuovi operatori
  - definire la rappresentazione dei nuovi dati in termini di dati pre-esistenti
  - contenere un insieme di sottoprogrammi che realizzino gli operatori definiti sui nuovi dati

Struttura di dati

- Come è noto, il tipo di dato è un modello matematico che consiste nella definizione di una collezione di valori (dominio) sui quali sono ammesse certe operazioni (funzioni)
- Una struttura di dati è un particolare tipo di dato, caratterizzato più dalla organizzazione imposta agli elementi che la compongono che dal tipo degli elementi componenti
- Le strutture solitamente disponibili nei linguaggi di programmazione sono le tabelle monodimensionali (vettori, array) i cui elementi sono contenuti in memoria centrale in locazioni contigue

Struttura di dati /2

- In generale, distinguiamo strutture
  - **lineari**: dati disposti in sequenza
    - Primo elemento, secondo elemento, …
  - **non lineari**: in cui non è individuata una sequenza
  - **a dimensione fissa (statiche)**: in cui il numero di elementi non può variare nel tempo
  - **a dimensione variabile (dinamiche)**: in cui il numero di elementi può variare nel tempo
  - **omogenee**: dati dello stesso tipo
  - **non omogenee**: dati non dello stesso tipo

Esempio

- La classica struttura vettore (array) è a dimensione fissa
- E' una tabella monodimensionale di elementi omogenei, su cui possono effettuarsi operazioni di
  - lettura (o selezione) reperimento del valore di un elemento
  - scrittura (o sostituzione) di un valore di un elemento con un nuovo valore
- Un vettore di n componenti è inteso come sequenza di oggetti in relazione d'ordine tra loro
  - L'organizzazione in memoria consente l’accesso diretto al singolo componente attraverso l'indice

Esempio /2

- Se volessimo definire le specifiche di un vettore
  - Specifica sintattica

    Tipi:
    vettore, intero, tipoelem

    Operatori:
    creavettore : ( ) → vettore
    leggivettore : (vettore, intero) → tipoelem
    scrivivettore: (vettore, intero, tipoelem) → vettore

Esempio /3

- Specifica semantica

  Tipi:
  intero: l'insieme dei numeri interi
  vettore: l'insieme delle sequenze di n elementi di tipo tipoelem

  Operatori:
  creavettore = v
  post: per ogni i, $1 \leq i \leq n$, l'i-esimo elemento del vettore v(i) è uguale ad un definito elemento di tipo tipoelem
  leggivettore(v, i) = e
  pre: $1 \leq i \leq n$
  post: e = v(i)
  scrivivettore(v, i, e) = v'
  pre: $1 \leq i \leq n$
  post: v'(i) = e, v'(j) = v(j) per ogni j tale che $1 \leq j \leq n$ e $j \neq i$

Esempio /4

- Realizzazione
  - in molti linguaggi il vettore è il tipo di dato primitivo
- In fortran
  - creavettore corrisponde a dimension v(n)
  - leggivettore(v, i) corrisponde a v(i)
  - scrivivettore (v, i,e) corrisponde a v(i)=e
- In pascal corrisponde all'array
  - la corrispondenza tra la specifica introdotta e quella del pascal è
    - creavettore corrisponde a var v : array[1..n] of tipoelem
    - leggivettore(v, i) corrisponde a v [i]
    - scrivivettore(v, i,e) corrisponde a v [i]:=e

Ulteriore Esempio

- La matrice, intesa come tabella a due o a molte dimensioni, è diretta estensione del vettore. Prevede due o più indici che consentono l'accesso diretto al singolo elemento.
- Per una matrice a tre dimensioni di n*m*p elementi diamo le specifiche
  - Specifica sintattica

    Tipi :
    matrice, intero, tipoelem

    Operatori:
    creamatrice: ( ) → matrice
    leggimatrice: (matrice, intero, intero, intero) → tipoelem
    scrivimatrice: (matrice, intero, intero, intero, tipoelem)
    → matrice

Ulteriore Esempio /2

- Specifica semantica

  Tipi :
  intero: insieme di interi
  matrice: insieme di n*m*p elementi di tipo tipoelem t.c. l'elemento caratterizzato dagli indici i, j e k è quello nella i-esima riga, j-esima colonna e k-esima dimensione in profondità

  etc. ….

Ulteriore Esempio /3

- Realizzazione
  - nella maggior parte dei linguaggi (c, pascal, …) è disponibile il dato primitivo matrice multidimensionale ed è riportato al dato primitivo array

    creamatrice corrisponde a
    var w : array[1..n,1..m,1..p] of tipoelem

    leggimatrice(w,i,j,k) corrisponde a
    w[i,j,k]

    scrivimatrice (w,i,j,k,e) corrisponde a
    w[i,j,k]:=e

Ulteriore Esempio /4

- Ma se volessimo realizzare, attraverso astrazione di dati e astrazione funzionale, proprio una struttura che rappresenti l'ente geometrico matrice e le operazioni che la matematica prevede (ad esempio il prodotto righe per colonne) come dovremmo operare?
- Che algebra dovremmo definire?
- Il pascal prevede gli array come dati primitivi, ma cosa fare per assicurare la astrazione corretta?

Ulteriore Esempio /5

- Nell'esempio che segue mostreremo come, pur essendo in grado di assicurare l'operazione di prodotto di matrici facendo uso di dati primitivi, non siamo in grado di garantire i requisiti di astrazione sulle strutture se non utilizziamo il concetto di tipo
- Esempio
  - il prodotto righe per colonne di due matrici A e B di dimensioni, rispettivamente (m x p) e (p x n), è una matrice C di dimensioni (m x n)

$$C(i,j) = \sum_{k=1}^{p} A(i,k)B(k,j)$$

Ulteriore Esempio /6

- Il calcolo è molto semplice

```
for i=1 to m do
    for j=1 to n do
        s = 0
        for k=1 to p do
            s = s + A(i,k) * B(k,j)
        C(i,j) = s
```

- Costituirà il corpo della procedura (operatore)
- Ma come utilizzare le dichiarative per realizzare l'astrazione?

Ulteriore Esempio /7

- Specifica sintattica
  - di matrice con l'aggiunta dell'operatore prodotto matriciale righe x colonne

    Tipi
    matrice , indice , tipoelem

    Operatore
    eseguiprodotto (A, B, m , n , p) → C

    dove:
    A è una matrice di dimensioni (m x p)
    B è una matrice di dimensioni (p x n)
    C è la matrice prodotto di dimensioni (m x n)

Ulteriore Esempio /8

- Specifica semantica
  - eseguiprodotto (A, B, m , n , p) → C

    PRE:
    A { A(i,k) t.c. i = 1,...,m e k = 1, ..., p }
    B { B(k,j) t.c. k = 1,...,p e j = 1, ..., n }

    POST:
    C { $C(i,j) = \sum_{k=1}^{p} A(i,k)B(k,j)$ t.c. i = 1,...,m e j = 1, ..., n }

Ulteriore Esempio /9

- Realizzazione
  - Una dichiarativa inadatta non usa i tipi
  - Definizione di costanti:
    - M = .. , P = .. ed N = ..
  - Definizione di variabili:
    - I intero in 1 . . M
    - J intero in 1 . . N
    - K intero in 1 . . P
    - S elemento di tipo TIPOELEM
    - ELEM elemento di tipo TIPOELEM
    - A array bidimensionale MxP di elementi di tipo TIPOELEM
    - B array bidimensionale PxN di elementi di tipo TIPOELEM
    - C array bidimensionale MxN di elementi di tipo TIPOELEM
    - ……..

Ulteriore Esempio /10

- Realizzazione
  - Una dichiarativa più opportuna usa i tipi
  - Definizione di costanti
    - DIMMAX = 50
  - Definizione di tipi
    - TIPOELEM : alias per un tipo di dato
    - INDICE : intero in 1..DIMMAX
    - MATRICE: array bidimensionale DIMMAX x DIMMAX di elementi di tipo TIPOELEM
  - Definizione di variabili
    - NUMERO_RIGHE_PRIMA, NUMERO_COLONNE_PRIMA, NUMERO_RIGHE_SECONDA, NUMERO_COLONNE_SECONDA: variabili di tipo INDICE
    - DIMCOM: variabile di tipo INDICE (*dimensione comune*)
    - PRIMAMAT,SECONDAMAT,MATPRODOTTO: variabili di tipo MATRICE
  –

Ulteriore Esempio /11

- Intestazione procedura ESEGUIPRODOTTO
  - ESEGUI_PRODOTTO (MATRICE A, MATRICE B, INDICE M, INDICE N, INDICE P, MATRICE per riferimento C)
    - Come si vede l'operatore è riservato a matrici definite omogenee (tipo MATRICE) ma si dovrà controllare nel programma che il numero di colonne della prima matrice sia uguale al numero di righe della seconda (DIMCOM)
- Chiamata procedura ESEGUI_PRODOTTO
  - ESEGUI_PRODOTTO (PRIMAMAT, SECONDAMAT, NUMERO_RIGHE_PRIMA, NUMERO_COLONNE_SECONDA, DIMCOM, MATPRODOTTO);

Tecniche di specifica

- Due tecniche comuni per la scrittura di specifiche sono
  - Specifiche Assiomatiche (o algebriche)
  - Modelli astratti (approccio costruttivo)
- In entrambe le tecniche la sintassi di ogni operatore viene specificata in termini del nome dell'operatore e dei tipi di parametri (stessa specifica sintattica)
- La differenza fra le due sta nel come viene definita la semantica degli operatori
- Le specifiche assiomatiche sono self-contained
  - Specificano ogni oggetto come composizione di funzioni
- I modelli astratti definiscono la semantica delle operazioni in termini di un altro ben definito tipo di dato
- La chiave di queste e di altre tecniche di specifica sta nel fatto che la descrizione della semantica non fa riferimento alla realizzazione

Specifica di un Dato Astratto

- Possiamo definire formalmente una specifica di un tipo di dato astratto come una tripla (D, F, A)
  - D: l'insieme di tutti i domini per la definizione del dato
    - Il tipo di dato che stiamo definendo, indicato con il simbolo d, è chiamato dominio designato; tutti gli altri domini che fanno parte del dato sono detti domini ausiliari.
      - Nel caso dello Stack, D potrebbe essere {Stack, ItemType, Boolean}
        - d = Stack — dominio designato
        - {ItemType, Boolean} — domini ausiliari
  - F: l'insieme degli operatori
    - Nel caso dello Stack: Create, Top, Push, Pop, IsEmpty
  - A: l'insieme degli assiomi o regole che descrivono la semantica degli operatori
    - Nel caso dello Stack dovrebbero definire la proprietà Last In First Out (LIFO) dello stack

Tipi di operatori

- Di base
  - **Costruttori** (Constructors): costruiscono/inizializzano una nuova istanza del dato
  - **Modificatori** (Transformers/Modifiers): operatori che cambiano la struttura in qualche modo
    - Potrebbero rendere la struttura vuota, aggiungere un elemento nella struttura, o rimuovere uno specifico oggetto dalla struttura
  - **Osservatori** (Observers/Accessors): permettono di osservare lo stato del dato senza modificarlo
    - Fanno domande true/false sul tipo di dato (la struttura è vuota?)
    - selezionano o accedono ad particolare oggetto (dammi la copia dell'ultimo elemento)
    - restituiscono una proprietà della struttura (quanto oggetti ci sono nella struttura?)
- Aggiuntivi
  - **Distruttori** (Destructors): puliscono l'occupazione di memoria del dato e la rilasciano per il riuso
  - **Iteratori** (Iterators): sono usati per permettere all'utente di processare l'intera struttura, componente per componente.
    - Avremo bisogno di operatori che inizializzano il processo di iterazione, e di operatori che ci restituiscono le “successive componenti”

Specifica assiomatica (o algebrica)

- Le specifiche assiomatiche definiscono il comportamento di un tipo di dato astratto dando gli assiomi che legano tra loro gli operatori
- Prima parte di una specifica algebrica per il tipo Stack

struttura Stack(di ItemType)

interfaccia

Create	→ Stack
Push(Stack, ItemType)	→ Stack
Pop(Stack)	→ Stack
Top(Stack)	→ ItemType
IsEmpty(Stack)	→ Boolean

d = Stack(di ItemType)
D = {Stack, ItemType, Boolean}
F = {Create, Push, Pop, Top, IsEmpty}

Specifica assiomatica (o algebrica) /2

- L'interfaccia elenca le funzioni, il tipo dei parametri (domini) e i tipi per i risultati (range)
  - Non c'è nessuna indicazione su cosa faccia ogni funzione
  - La relazione fra lo stack di input e quello di output nella funzione Pop, ad esempio, non è specificato
  - La semantica (il significato) degli operatori viene dato con gli assiomi
- Ogni istanza del dato astratto Stack può essere denotata come una sequenza di applicazioni di operatori detta espressione
  - Push(Push(Push(Create, 1), 2), 3)
- Gli operatori observer non fanno parte di una espressione
  - Vengono applicati ad una espressione, non modificano la struttura

Specifica assiomatica (o algebrica) /3

- Gli assiomi costituiscono la terza componente della nostra tripla
  1) Gli assiomi per gli observer sono scritti in termini dei costruttori
  2) I costruttori che rimuovono un elemento dalla struttura sono definiti in termini dei costruttori che aggiungono un elemento alla struttura
  3) I costruttori che creano la struttura o aggiungono un elemento sono sono definiti esplicitamente

Per la struttura Stack avremo:

assiomi **per ogni S in Stack, i in ItemType:**

IsEmpty(Create) = True
IsEmpty(Push(S,i)) = False
Top(Create) = Error
Top(Push(S,i)) = i
Pop(Create) = Error
Pop(Push(S,i)) = S

Possiamo vedere Error come un elemento comune a tutti i domini (assume si un valore ItemType che un valore Stack)

Specifica assiomatica (o algebrica) /4

- Gli assiomi vanno interpretati nel seguente modo
  - Dato un operatore su uno stack S, si cerca fra gli assiomi quello la cui parte sinistra è confrontabile con l'espressione S e se ne restituisce la parte destra
    - IsEmpty(S)
      - Se S è vuoto (operatore Create) si restituisce True
      - Se S non è vuoto (effettuato un Push su S) si restituisce False
- top e pop non sono definiti per stack vuoti
  - Si dice che alcune funzioni sono **parziali** (-/>) e non totali (->)
    - Non definite per uno o più elementi del dominio
  - Possiamo restringere l'applicabilità degli operatori al proprio dominio inserendo delle precondizioni
    - Precondizioni per pop(Stack S) e top(Stack S)
      - not empty(S)

Specifica costruttiva (o modello astratto)

- **Specifica sintattica**

  struttura Stack(di ItemType)

  interfaccia
  Create → Stack
  Push(Stack, ItemType) → Stack
  Pop(Stack) → Stack
  Top(Stack) → ItemType
  IsEmpty(Stack) → Boolean

- d = Stack(di ItemType)
  - L'insieme di tutte le istanze di stack
- ItemType
  - L'insieme di tutti gli item che possono essere presenti in uno stack
- Boolean
  - L'insieme dei valori di verità true e false

Specifica costruttiva (o modello astratto) /2

- **Specifica semantica**
  - In questo approccio diamo la semantica di ogni operatore associando ad ognuno una precondizione ed una postcondizione
  - Precondizione
    - Asserzione logica che specifica le caratteristiche richieste dei valori dei parametri
  - Postcondizione
    - Asserzione logica che specifica le caratteristiche del risultato rispetto agli argomenti

Specifica costruttiva (o modello astratto) /3

- Push(S, I) → S'
  - PRE: -
  - POST: S' è lo stack S con I al top
- Pop(S) → S'
  - PRE: not empty(S)
  - POST: S' è lo stack S con l'elemento top eliminato
- Top(S) → I
  - PRE: not empty(S)
  - POST: I è l'elemento top dello stack S