Specifica – parte IIIA

Leggere Sez. 5.6 Ghezzi et al.

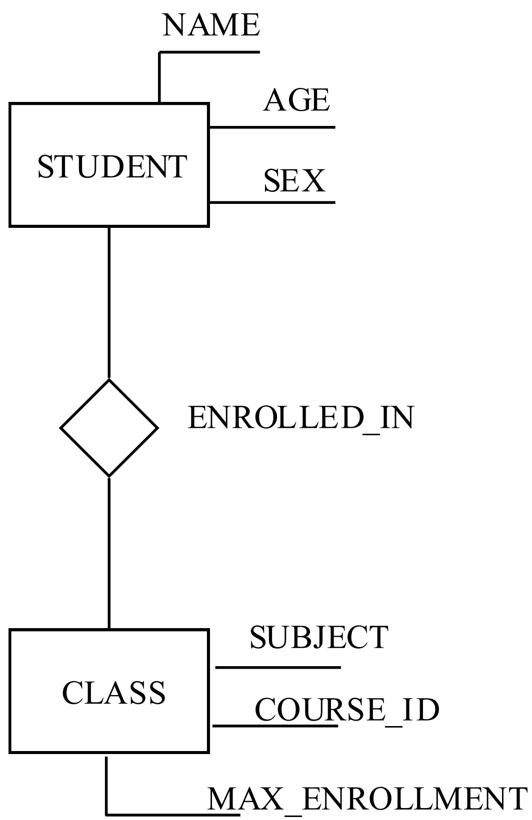
Specifiche descrittive

- Specificano i sistemi (o parti di essi) attraverso le loro *proprietà* anziché il loro *comportamento*.
- Diagrammi entità/relazione (semi-formale)
- Specifiche logiche
- Specifiche algebriche

Diagrammi Entità-Relazione

- Modello concettuale dei dati
- Si possono usare come complemento dei DFD
- Concetti primitivi:
  - Entità
  - Relazione
  - Attributo

Esempio


NAME
AGE
STUDENT
SEX
ENROLLED_IN
SUBJECT
CLASS
COURSE_ID
MAX_ENROLLMENT

Diagrammi E/R

- Entità: collezione di oggetti che condividono proprietà comuni.
- Concetto simile al tipo nei linguaggi di programmazione
- Rappresentate da rettangoli
- Proprietà:
  - Attributi (elenco vicino all'entità)
  - Relazioni (rombo)

Versioni dei linguaggi E/R

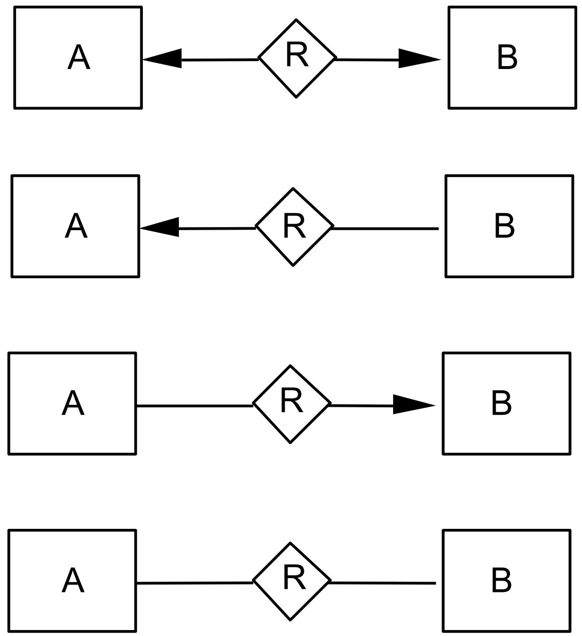
- Linguaggio non standardizzato
- In alcune versioni relazioni solo binarie, in altri n-arie.
- Alcuni linguaggi permettono di associare attributi alle relazioni (es. PROFICIENCY associato a ENROLLED_IN).

Versioni dei linguaggi E/R

- Alcuni linguaggi permettono la relazione IS_A (una sorta di ereditarietà). Es: UNDERGRADUATE IS_A STUDENT (eredita le proprietà – attributi e relazioni - e può aggiungerne di nuove)
- Alcuni linguaggi permettono *relazioni parziali*: non tutti gli elementi delle entità coinvolte devono partecipare alla relazione.

Molteplicità delle relazioni

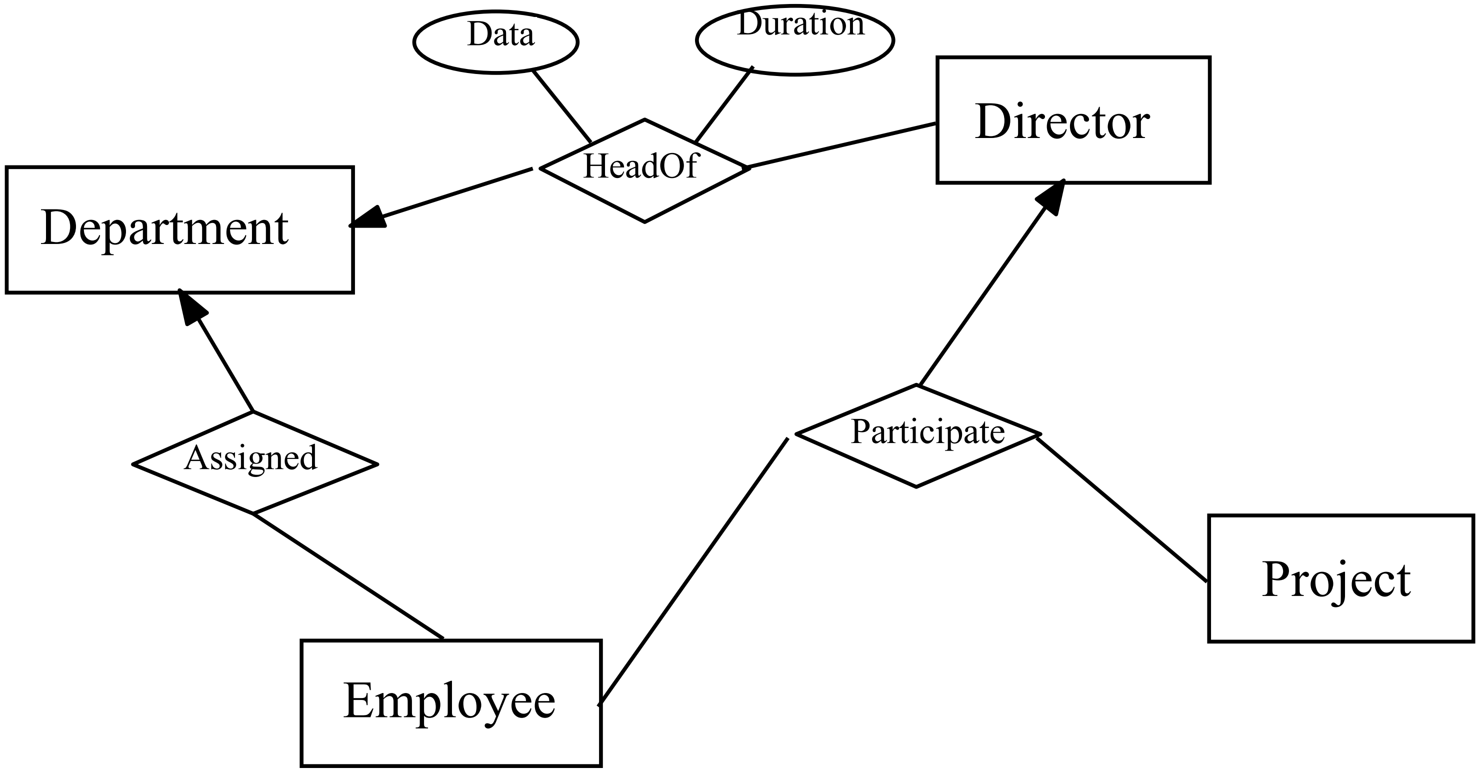
- Si possono imporre molteplicità alle relazioni:
  - one-to-one
  - one-to-many
  - many-to-one
  - many-to-many



Molteplicità: A r B

- one-to-one:
  - $\forall a,a' \in A, \forall b \in B, a\ r\ b, a'\ r\ b \rightarrow a = a'$
  - $\forall a \in A, \forall b,b' \in B, a\ r\ b, a\ r\ b' \rightarrow b = b'$
- one-to-many
  - $\forall a,a' \in A, \forall b \in B, a\ r\ b, a'\ r\ b \rightarrow a = a'$
- many-to-one
  - $\forall a \in A, \forall b,b' \in B, a\ r\ b, a\ r\ b' \rightarrow b = b'$
- many-to-many

Relazioni non binarie



Notazione semi-formale

- Molte restrizioni non sono esprimibili
- Ad esempio: una classe esiste solo se ha più di 5 iscritti e meno di MAX_ENROLLMENT iscritti (attributo di CLASS)
- Queste restrizioni possono essere aggiunte come commenti (da cui la semi-formalità)

Specifiche logiche

- Specificano le proprietà del programma per mezzo di *formule logiche*.
- La logica classica si suddivide in:
  - logica proposizionale (formule senza variabili)
  - logica dei predicati (in cui le formule possono contenere variabili e quantificatori)

Logica del primo ordine (o dei predicati)

- *Alfabeto* composto da:
  - insieme C dei *simboli di costante*
  - insieme F dei *simboli di funzione*
  - insieme P dei *simboli di predicato*
  - insieme V dei *simboli di variabile*
  - *connettori logici* **and**, **or**, **not**, **implies**, $\equiv$
  - *parentesi*
  - *quantificatori* $\forall$ (universale), $\exists$ (esistenziale)

Sintassi

- Termini elementari: variabili, costanti, applicazioni di funzioni (f(...)) a termini elementari
- Esempi:
  - costante c_0
  - variabile x
  - se f e g sono simboli di funzione: $f(x)$, $g(f(c_0),x)$

Sintassi

- Formule atomiche: applicazione di simboli di predicato (p(...)) a termini elementari
- Esempi: se p è un simbolo di predicato,
  - $p(x)$
  - $p(x,f(c_0))$
- Alcuni predicati e funzioni hanno sintassi particolare: es. $<$

Sintassi

- Formule ben formate (fbf):
  - le formule atomiche sono fbf
  - l'applicazione usuale dei connettori logici a fbf è una fbf
  - se A è una fbf e X è una variabile:
    - $\forall X A$
    - $\exists X A$

    è una fbf

Sintassi

- Esempi di formule ben formate:
  - $p(x)$
  - $p(x)$ **and** $p(f(c_0))$
  - $\forall x\ p(x)$
  - $\forall x\ \exists y\ (p(x) \text{ and } p(f(y)))$

Variabili

- Una variabile è detta
  - *libera* se non è quantificata
  - *legata* altrimenti
- Una formula senza variabili libere è detta *chiusa*.
- La *chiusura* di una formula si ottiene quantificando universalmente tutte le variabili libere

Semantica

- Una *interpretazione* associa un significato ai simboli di costante, funzione, predicato
- In questo modo ogni fbf assume un valore di verità, uguale a quello della sua interpretazione nel dominio del discorso
- Assegneremo ai simboli i significati consueti

Esempi

1. x > y **and** y > z **implies** x > z
2. x = y ≡ y = x
3. **for all** x, y, z (x > y **and** y > z **implies** x > z) *(chiusura di 1)*
4. x + 1 < x – 1
5. **for all** x (**exists** y (y = x + z))
6. x > 3 **or** x < -6

Specifica di programmi completli

- Precondizione: formula logica avente come variabili libere gli input del programma

  {Pre (i1, i2,..., in) }

- Postcondizione: formula logica avente come variabili libere gli input e gli output del programma

  {Post (o1, o2,..., om, i1, i2,..., in)}

Asserzioni input-output

- Dato un programma P, la *proprietà* o *requisito*

  $\{Pre\ (i_1, i_2, \ldots, i_n)\ \}$

  P

  $\{Post\ (o_1, o_2, \ldots, o_m, i_1, i_2, \ldots, i_n)\}$

  significa che se la precondizione è vera prima dell'esecuzione del programma, la postcondizione sarà vera dopo l'esecuzione

Esempi (divisione, massimo)

- $\{$**exists** $z\ (i_1 = z^*i_2)\}$
  P
  $\{o_1 = i_1/i_2\}$ *(divisione)*
- $\{i_1 > i_2\}$
  P
  $\{i_1 = i_2 * o_1 + o_2$ **and** $o_2 >= 0$ **and** $o_2 < i_2\}$
- $\{true\}$
  P
  $\{o = i_1$ **or** $o = i_2\}$ **and** $o >= i_1$ **and** $o >= i_2\}$

Esempi

- $\{n > 0\}$ *(sequenza lunga n, somma)*
  P
  $\{o = \Sigma_{k=1..n}\ i_k\}$

- $\{n > 0\}$ *(calcola sequenza inversa)*
  P
  {**for all** i (1 <= i <= n) **implies** ($o_i = i_{n-i+1}$)}

Esempi

$\{i_1 > 0$ **and** $i_2 > 0\}$ *(MCD)*

P

$\{($**exists** $z_1, z_2\ (i_1 = o * z_1$ **and** $i_2 = o * z_2)$

and not (**exists** h (**exists** $z_1, z_2\ (i_1 = h * z_1$ **and** $i_2 = h * z_2)$ **and** $h > o))\}$

Specifica di frammenti di programma

- *Asserzioni intermedie*: si riferiscono alle variabili del programma.
- {n > 0} -- n is a constant value

  procedure search (table: in integer_array; n: in integer; element: in integer; *found*: out Boolean);

  {*found* $\equiv$ (**exists** i (1 $\leq$ i $\leq$ n **and** table (i) = element))}

Asserzioni intermedie

- {n > 0 }
  procedure reverse (a: **in out** integer_array; n: **in** integer);
  {**for all** i (1≤ i ≤ n) **implies** (a (i) = *old*_a (n - i +1))}
- {n > 0}
  procedure sort (a: **in out** integer_array; n: **in** integer)
  {sorted(a, n)}
  dove sorted(a, n)≡(**for all** i(1<=i<=n) **implies** a(i) <= a(i+1))

Specifica di classi

- *Caratterizzazione logica* degli stati degli oggetti e delle operazioni possibili
- Predicati *invarianti*: proprietà che caratterizzano gli stati lungo tutta la vita dell'oggetto
- *Precondizioni* e *postcondizioni* delle operazioni

Esempio

- Invariante per il tipo di dato astratto INSIEME (implementato usando un array IMPL di lunghezza length):

  for all i, j ($1 \leq i \leq$ length and $1 \leq j \leq$ length and $i \neq j$) **implies** $IMPL[i] \neq IMPL[j]$

  (cioè non ci sono elementi duplicati)

Esempio

- Precondizione dell' operazione DELETE

  exists i (1<= i <= length **and** IMPL[i] = x)

- Postcondizione:

  for all i (1<= i <=length **implies** IMPL[i] != x) and **forall** i ((1 <= i <= old_length **and** old_IMPL[i] != x) **implies exists** j (1 <= i <= length **and** IMPL[j] = old_IMPL[i]))

Specifica completa

- INV invariante
- per ogni operazione op_i, pre_i e $post_i$ pre- e post-condizione per op_i
- {INV **and** pre_i} frammento per op_i {INV **and** $post_i$}
- {true} costruttore {INV}

- Prove formali di correttezza

Logica di Hoare: correttezza

- Associa ad ogni programma di un linguaggio L la ***relazione calcolata dal programma*** (formula logica)
- Programma Prog che termina sull'ingresso i e produce l'output o:

$$R_{Prog}: \qquad R_{Prog}(i,o)$$

- viene generalmente espressa nel calcolo dei predicati e lega ***pre-*** e ***post-condizione***

Metodo assiomatico - correttezza

- Se la precondizione P è vera sui dati di ingresso i ed il programma Prog termina, allora la postcondizione è vera sui dati di uscita o

  {P} Prog {Q} *specifica*

- Si dimostra che per ogni insieme dei dati di ingresso che soddisfa P, Prog termina ed i dati di uscita soddisfano Q

Specifica di comportamenti che non terminano

- Sistemi reattivi: attendono input e producono output, senza terminare
- Le sequenze di input e output possono non essere finite
- E’ sufficiente imporre condizioni
  - sulle sequenze parziali
  - in punti di esecuzione critici

Esempio

- Produttore, consumatore, buffer
- Invariante

  input_sequence = append (output_sequence, contents(CHAR_BUFFER))
- cioè quanto è stato scritto è la concatenazione di quanto è stato letto e del contenuto del buffer (al di fuori di operazioni del monitor, ovvero all'ingresso e all'uscita da tali operazioni)