I modelli descrittivi

- Le specifiche operazionali non rappresentano la logica di funzionamento e le proprietà complessive del sistema modellato

- I metodi di specifica descrittivi partono da una descrizione più astratta e svincolata dal riferimento a macchine o dispositivi

- Danno una descrizione degli ***stati ammissibili*** per il sistema modellato in modo più generale, mediante l'impiego di concetti matematici (equazioni, assiomi, vincoli e proprietà)

- Formalismi di tipo algebrico o logico

Logica

- Strumento per formalizzare il ragionamento umano
- Logica classica, si suddivide in *logica proposizionale* e *logica dei predicati*.
- Diverso grado di espressività: mentre nella logica dei predicati è possibile esprimere variabili e quantificazioni questo non è possibile nella logica proposizionale
- Alfabeto della logica dei predicati, consiste di cinque insiemi:
  - l'insieme dei simboli di costante, C;
  - l'insieme dei simboli di funzione, F;
  - l'insieme dei simboli di predicato (o relazione), P;
  - l'insieme dei simboli di variabile, V;
  - i connettivi logici:

    ~ (negazione), (congiunzione),

    Δ (disgiunzione), $\varnothing$ (implicazione),

    $\times$ (equivalenza),
  - le parentesi "(" ")"

- i quantificatori esistenziale ($\exists$) e universale ($\forall$)

Esempio

parente(giovanna, maria) (E1)

∃X (uomo(X) felice(X)) (E2)

∀X (uomo(X) ∅ mortale(X)) (E3)

∃X (uomo(X)) (E4)

∃X (uomo(f(X) (E5)

- *Formule ben formate* (sintatticamente corrette), si ottengono attraverso combinazione di formule elementari, dette formule atomiche, utilizzando i connettivi ed i quantificatori

- *Formule atomiche* sono ottenute applicando i simboli di predicato a termini elementari (variabili, costanti o applicazioni di funzioni $f(t_1,...,t_n)$)

- Un *atomo* o formula atomica è l'applicazione di un simbolo di predicato n-ario p ad n termini $t_1,...,t_n$: $p(t_1,..,t_n)$. Nell'esempio (E1) è una formula atomica.

Formule ben formate

- Dato l'alfabeto, le *formule ben formate* (*fbf*) del linguaggio, cioè le sole formule significative del linguaggio, sono definite ricorsivamente come segue:
  - ogni atomo è una fbf;
  - se A e B sono fbf, allora lo sono anche ~A, A B, AΔB, A$\varnothing$B, A$\times$B (eventualmente racchiuse tra parentesi tonde bilanciate);
  - se A è una fbf ed X è una variabile, $\forall$X A ed $\exists$X A sono fbf.

  Le espressioni (E1), (E2), (E3) sono formule ben formate, mentre non lo sono (E4) ed (E5)

Precedenza tra gli operatori:

~ $\exists$ $\forall$

Δ

$\varnothing$ $\times$

Interpretazione

- Associa un significato ai simboli.

- I simboli del sistema formale assumono in questo modo un significato.

- Ogni formula atomica o composta della logica dei predicati del primo ordine, in questo modo, può assumere il valore vero o falso in base alla frase che rappresenta nel dominio del discorso.

- **Esempio:**

  Quando scriviamo:

  $$\forall X \, \forall Y \, \forall Z \;\; (op(X, Y, Z) \varnothing op(Y, X, Z))$$

  se le variabili variano sull'insieme dei numeri reali tale formula è vera se il simbolo di predicato “op” ha il significato di un operatore commutativo (quale ad esempio la somma o la moltiplicazione), ma falsa se l'operatore non è commutativo (come ad esempio la sottrazione o la divisione)

- Più formalmente, dato un linguaggio del primo ordine L, un'*interpretazione* I per L definisce un dominio non vuoto D ed assegna:
  - a ciascun simbolo di costante in C una costante in D;
  - a ciascun simbolo di funzione n-ario in una funzione $F:D^n \varnothing D$;
  - a ciascun simbolo di predicato n-ario in P una relazione in D^n, cioè un sottoinsieme di D^n.

- Data un'interpretazione I:
  - Una formula atomica priva di variabili ha valore *vero* in I quando il corrispondente predicato è soddisfatto (cioè quando la corrispondente relazione è vera nel dominio). La formula atomica ha valore *falso* quando il corrispondente predicato non è soddisfatto.
  - Una formula composta ha un valore di verita` rispetto ad I che si ottiene da quello delle sue componenti utilizzando le tavole di verità dei connettivi logici:

A	B	~A	A ∧ B	A ∨ B	A ⇒ B	A ⇔ B
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

Dimostrazione

- Le ***regole di inferenza*** permettono di derivare fbf da fbf mediante ***trasformazioni sintattiche***

- Ciascuna regola di inferenza viene rappresentata nel modo seguente:

$$\frac{Condizioni}{Conclusione}$$

  dove la formula che si trova inferiormente è la conclusione che viene derivata a partire dalle condizioni o premesse della regola (le formule che si trovano superiormente)

- Le due regole di inferenza del calcolo dei predicati del primo ordine sono:

  Modus Ponens (MP):

$$\frac{A, A \rightarrow B}{B}$$

  Specializzazione (Spec):

$$\frac{\forall X \, \mathrm{A}}{A(t)}$$

Dimostrazione (cont.)

- Una *dimostrazione* è una sequenza finita di fbf f_1, f_2, ..., f_n, tale che ciascuna f_i o è un assioma oppure è ricavabile dalle fbf precedenti mediante una regola di inferenza

- L'ultima fbf di ogni dimostrazione è detta *teorema*, mentre la sequenza di regole di inferenza applicate è detta prova del teorema

- Una fbf F è *derivabile* in una teoria del primo ordine T (e scriveremo T $\vdash$F) se esiste una dimostrazione per F

- Una teoria per la quale esiste un metodo meccanico per stabilire se una qualunque fbf è un teorema o non lo è si dice *decidibile*

- Il calcolo dei predicati del primo ordine è *semi-decidibile* (la terminazione e` garantita solo se la fbf è un teorema)

Esempio

- Teoria che rappresenta la relazione di minore uguale sui numeri naturali:

p(0,0)	(A1)
$\forall X \forall Y (p(X,Y) \varnothing p(X,s(Y)))$	(A2)
$\forall X\ p(X,X)$	(A3)

- T o p(0,s(0)) secondo le seguenti trasformazioni:

  - da Spec e A2:

$\forall X \forall Y (p(X,Y) \varnothing p(X,s(Y))) \varnothing$ $\forall Y (p(0,Y) \varnothing p(0,s(Y)))$	(T1)

  - applicando MP a T1 e A2:

$\forall Y (p(0,Y) \varnothing p(0,s(Y)))$	(T2)

  - da Spec e T2:

$\forall Y (p(0,Y) \varnothing p(0,s(Y))) \varnothing (p(0,0) \varnothing p(0,s(0)))$	(T3)

- applicando MP a T3 e T2:

p(0,0)∅p(0,s(0))	(T4)

- applicando MP a T4 e A1:

p(0,s(0))	(T5)

La logica proposizionale

- Il *calcolo proposizionale*, o logica proposizionale, può essere formulato come sistema formale

- I simboli primitivi di questo sistema formale sono:
  - l'insieme dei simboli di proposizione A, B, C, ...;
  - i connettivi logici ~ (negazione), (congiunzione), Δ (disgiunzione), $\varnothing$ (implicazione), $\times$ (equivalenza);
  - le parentesi ( ).

- Le formule ben formate (fbf) del linguaggio sono definite ricorsivamente come segue:

  (1) Qualunque simbolo di proposizione è una fbf;

  (2) Se A e B sono fbf allora anche

  ~A A BAΔBA$\varnothing$B A$\times$B

  sono fbf.

Applicazioni della logica del primo ordine

- Utilizzata nella ***specifica*** dei programmi e nella ***verifica*** della loro correttezza

- E` adatta a descrivere le trasformazioni effettuate sui dati e a definire un programma come una funzione che calcola un certo valore a partire da certi dati forniti come ingresso

Specifica: esempio ordinamento

- La specifica dell'operazione viene espressa mediante un predicato *versioneOrdinata* che mette in relazione un vettore da ordinare con la sua versione ordinata

- Un vettore è la versione ordinata di un altro se ne costituisce una permutazione e se è di per sè ordinato:

versioneOrdinata (arrDis, arrOrd)

$\times$

permutazione (arrDis, arrOrd) ordinato (arrOrd)

- Due vettori (composti per ipotesi da *n* elementi *distinti*) sono uno la permutazione dell'altro se ogni elemento dell'uno è anche incluso, in una posizione qualsiasi, nell'altro:

permutazione (a, b)

$$\times$$

$$\forall k\ (1=k=n \varnothing \exists i \quad (1=i=n \quad b[i]=a[k])$$
$$\exists j\ (1=j=n \quad a[j]=b[k])\ )$$

- Un vettore è ordinato se i suoi elementi sono disposti in ordine crescente:

$$\text{ordinato}\ (a) \times \forall k\ (1=k<n \varnothing a[k] < a[k+1]\ )$$

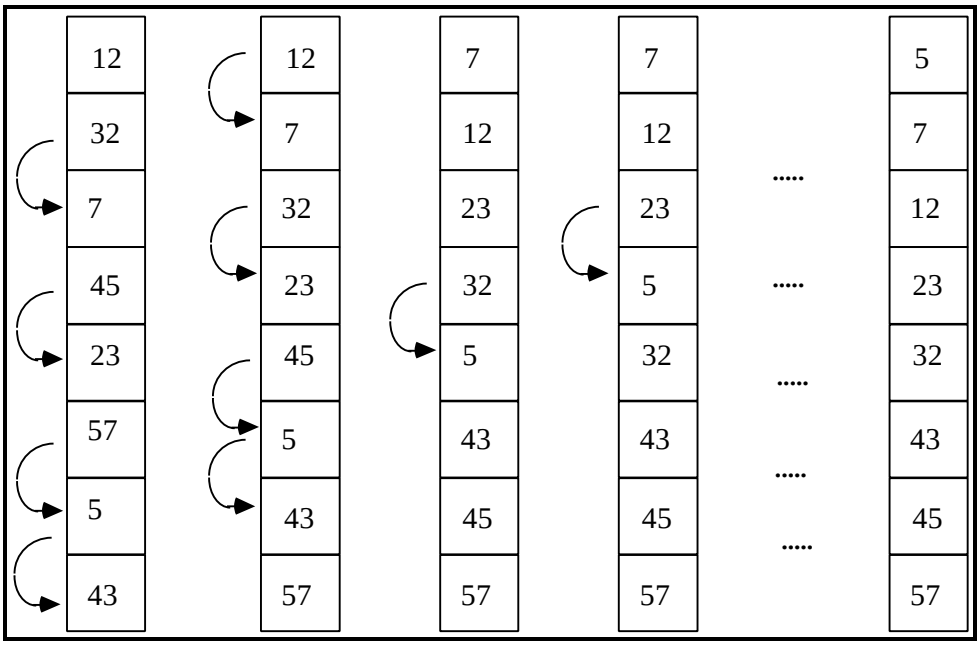
Specifica (eseguibile) in Prolog

```
ordina(Disordinato,Ordinato) :-
    permutation(Disordinato, Ordinato),
    ordinato(Ordinato).

permutation([],[]).
permutation(X,[Y|Z]):-select(Y,X,Xresto),
                      permutation(Xresto,Z).

ordinato([]).
ordinato([El]).
ordinato([El1,El2|Resto]):-
               El1<El2,
               ordinato([El2|Resto]).

select(Y,[Y|Xresto],Xresto).
select(Y,[Y1|Resto],[Y1|Resto1):-
               select(Y,Resto,Resto1).
```

Correttezza: esempio ordinamento

- Programma di ordinamento mediante l'algoritmo bubblesort:

```
type vettore=: array [1..n] of integer;

procedure BubbleSort (var vett: vettore);
var   i,j: 1..n;  temp : integer;
begin
      for i:=1 to n-1 do
      for j:=1 to n-i do
          if vett[j] > vett[j+1]
          then temp := vett [j];
                    vett [j] := vett [j+1];
                    vett [j+1] := temp;
          end; {end if}
      {end for:"k(n-i=k<n ÿ$vett[k]=vett[k+1]) }
  {end for:"k (1=k<n ÿ$vett[k]=vett[k+1]) }
end;   {end BubbleSort}
```


12
32
7
45
23
57
5
43
12
7
32
23
45
5
43
57
7
12
23
32
5
43
45
57
7
12
23
5
32
43
45
57
.....
.....
.....
.....
.....
5
7
12
23
32
43
45
57

Logica di Hoare: correttezza

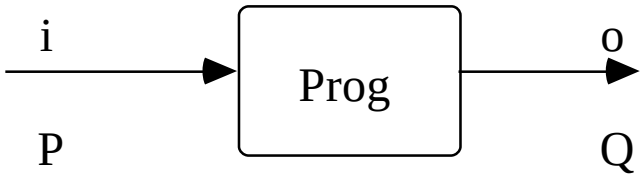
- Associa ad ogni programma di un linguaggio L la ***relazione calcolata dal programma*** (formula logica)

- Programma P che termina sull'ingresso i e produce l'output o:

$$R_P: R_P(i,o)$$

- R_P viene generalmente espressa nel calcolo dei predicati

- Lega ***pre-*** e ***post-condizione***:



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

- Semantica assiomatica di Prog, coppia P, Q tale che vale l'espressione:

$$\{P\}\ Prog\ \{Q\}$$

Metodo assiomatico

- Il metodo assiomatico viene utilizzato per eseguire ***prove formali*** della correttezza di un programma
- Per ***ciascuna istruzione*** del linguaggio occorre definire come sono correlate pre- e post-condizioni
- Specificato attraverso ***assiomi*** o ***regole di inferenza***
- Dimostrare che Prog e` ***corretto*** rispetto alla specifica:

{P} Prog {Q} ***specifica***

  significa dimostrare che per ogni insieme dei dati di ingresso che soddisfa P, Prog termina ed i dati di uscita soddisfano Q

- Q e` ***derivabile*** a partire da P, usando assiomi e regole di inferenza che definiscono la semantica del linguaggio in cui e` scritto il programma Prog

Esempio (mini-Pascal):

{P} **readln**(n) {P[i/n]}

{P} **writeln**(n) {P}

{P} n**:=**exp {P[v/n]}

dove v e` il valore risultante dalla valutazione di exp

$$\frac{\{P_1\}\ i_1\ \{P_2\} \qquad \{P_2\}\ i_2\ \{P_3\}}{\{P_1\}\ i_1;\ i_2\ \{P_3\}}$$

$$\frac{\{P_1\ \ v(bool)=true\}\ i_1\ \{P_2\} \qquad \{P_1\ \ v(bool)=false\}\ i_2\ \{P_2\}}{\{P_1\}\ \textbf{if}\ bool\ \textbf{then}\ i_1\ \textbf{else}\ i_2\ \{P_2\}}$$

$$\frac{\{P\ \ v(bool)=true\}\ i\ \{P\}}{\{P\}\ \textbf{while}\ bool\ \textbf{do}\ i\ \{P\ \ v(bool)=false\}}$$

- Se P e` vera prima di eseguire l'istruzione while e questa termina, P e` vera anche dopo (ed a questo punto bool ha valore falso):***invariante del ciclo***.

Prove formali di correttezza: esempio

- Calcolo del modulo di due numeri interi non negativi: r=x **mod** y (usando solo +, -, *).

  P: x>=0 **and** y>0
  Q: deve esistere un intero q:
  x=y*q+r **and** 0<=r<y

```
program modulo (input, output);
var x,y,q: integer;
begin
   readln(x,y);                    {P: x>=0 and y>0}
   q:=0;
   r:=x;                           {x=y*q+r  and r>=0}
   while r>=y do
      begin
         r:=r-y;
         q:=q+1
      end;
   writeln(r)                      {Q: x=y*q+r  and 0<=r<y}
end.
```

```
readln(x,y);                    {P: x>=0 and y>0}

begin
q:=0;
r:=x;
end;                            {W: x=y*q+r  and r>=0}

{W}
while r>y do
   begin
        r:=r-y;
        q:=q+1
   end;
{Q:     x=y*q+r  and  0<=r<y}
```

Dimostriamo che W e` l'invariante del ciclo:

*{W: x=y*q+r and r>=0}*
while r>=y **do** *{W and r>=y}*
begin
r:=r-y;
q:=q+1
end;
{W and r<y}

Dopo una iterazione:

r':=r-y;
q':=q+1;
x':=x;
y':=y
*{W': x'=y'*q'+r' and r'>=0}*

e` vera, infatti:
r'>=0, perche` si e` decrementato r di y e si aveva r>=y
x'=y'*q'+r', perche` si e` incrementato q e decrementato r di y.

All'uscita dal ciclo **while:**

*{W and r<y} = {x=y*q+r and 0<=r<y} = Q*

Logica temporale

- Estensione della logica classica
- Impiegata nella specifica di ***sistemi concorrenti*** e in ***tempo reale***
- Aspetti *temporali* di importanza preponderante rispetto agli aspetti di trasformazione dei dati e calcolo di valori in uscita
- Aggiunge costrutti per esprimere formalmente e in modo naturale i concetti temporali e mantiene un riferimento costante all'istante corrente
- La verità di una formula diventa un ***concetto dinamico***, che varia da istante ad istante
- Aggiunge operatori che fungono da quantificatori rispetto al dominio temporale:

possibilità	(◇)

necessità	(□)

  nel tempo, futuro o passato

- Basata sulla *logica modale*

Logica modale

- Definita da:

<W, R, V>

W rappresenta l'insieme dei mondi

R Õ W¥ W è la relazione chiamata di *raggiungibilità* tra i mondi

V : F ¥ W Æ {false, true}, funzione di valutazione delle formule dove *F* è l'insieme delle formule della teoria modale

- Due nuovi operatori, ◇ e □, che esprimono la possibilità e la necessità delle formule loro argomento nell'insieme dei mondi raggiungibili a partire da quello nel quale viene valutata la formula complessiva

Logica modale (cont.)

- Valore di verita` in una logica modale:

V (◇f, w) = true

se e solo se ∃v W tale che w R v e V(f, v) = true;

V (□f, w) = true

se e solo se ∀v W tale che w R v, V(f, v) = true

- La formula ◇*f* è vera in un mondo *w* se e solo se esiste un mondo *v* raggiungibile da *w* nel quale *f*, l'argomento dell'operatore temporale, è vera

- La formula □*f* è invece vera in *w* se e solo se, in tutti i mondi raggiungibili da *w*, l'argomento *f* è vera

- Operatori modali ◇ e □ come quantificatori definiti sull'insieme dei mondi raggiungibili da quello corrente (◇ esistenziale, □ universale)

Logica temporale (cont.)

- ***Significato temporale:*** interpretazione di *R* come relazione *prossimo istante* (mondi come insieme delle configurazioni che il sistema modellato può assumere in istanti successivi di tempo)

- Applicazione allo ***studio delle proprietà temporali*** dei sistemi (*logica temporale*)

- Logiche temporali diverse a seconda della struttura del *dominio temporale* cui fanno riferimento
  - tempo finito o infinito, sia nel futuro che nel passato;
  - tempo discreto, denso o continuo;
  - con struttura lineare, con diramazioni e ri-congiungimenti nel futuro e/o nel passato, o circolare

Logica temporale lineare

- Utilizzata per la specifica del comportamento temporale dei sistemi concorrenti (logica temporale lineare proposizionale)

- Dominio temporale di tipo ***discreto***, con programmi rappresentati mediante grafi molto simili agli automi finiti

- Dominio temporale costituito da una sequenza discreta di *stati*,

  $s = s0, s1, s2, s3, \ldots$

  che possiede uno stato iniziale *s0* ed è infinita in una direzione

- Operatori temporali:
  - *X* (Next, cioè nel prossimo stato),
  - *F* (Future, cioè in uno stato del futuro),
  - *G* (sempre nel futuro) e
  - *U* (Until, cioè fino a ...).

Logica temporale lineare (cont.)

- Se $s \models A$ indica il fatto che la formula A e` vera nella sequenza $s = s_1, s_2, s_3, \ldots$, allora:

$\sigma \models p$	con p variabile proposizionale, se e solo se p è vera in s_0
$\sigma \models \neg A$	se e solo se σ? A
$\sigma \models A \varnothing B$	se e solo se σ σ? A, oppure $\sigma \models B$
$\sigma \models X\ A$	se e solo se $\sigma_1 \models A$,
$\sigma \models F\ A$	se e solo se esiste un $i=0$ tale che $\sigma_i \models A$
$\sigma \models G\ A$	se e solo per ogni $i=0$ si ha $\sigma_i \models A$
$\sigma \models A\ U\ B$	se e solo se esiste un j=0 tale che $\sigma_j \models B$, e per ogni i con $0=i<j$ si ha $\sigma_i \models A$

Logica temporale lineare (cont.)

- Significato intuitivo degli operatori temporali:

  - una formula *A* è vera in una sequenza *s* se è vera nel suo primo stato;

  - la formula *XA* è vera se *A* è vera nel prossimo stato della sequenza

  - *FA* è vera se esiste uno stato futuro della sequenza in cui la formula *A* è vera

  - *GA* è vera se *A* è vera in tutti gli stati futuri

  - *A U B* è vera se *B* si verifica in uno stato futuro e in tutti gli stati intermedi, dal presente incluso fino a quello escluso, *A* si verifica

- La logica temporale lineare è stata usata per esprimere e dimostrare le proprietà dei programmi concorrenti

Proprietà

sicurezza

Chiamata anche *invarianza* (in inglese, *safety*):

Espressa mediante formule del tipo *Gq*, oppure *pÆ Gq*, che indicano che la condizione *q* richiesta risulta sempre vera, eventualmente condizionata al verificarsi, inizialmente, della condizione *p*

Proprietà di questo tipo significano che una data configurazione o condizione del sistema specificato viene mantenuta permanentemente

Ad esempio, l'assenza di errori dipendenti dal tempo nei sistemi reattivi, o la *mutua esclusione* nell'accesso a sezioni critiche dei programmi

vitalità

Chiamata anche *necessità* (in inglese, *liveness*)

Assicura che una condizione o evento desiderato si verificherà nel futuro

Proprietà di questo tipo sono espresse da formule del tipo *Fq* oppure *pÆ Fq*, che indicano che la condizione *q* si verificherà nel futuro, eventualmente sotto la condizione presente di *p*

Ad esempio, proprietà di *terminazione*, proprietà di *raggiungibilità* di parti dei programmi,

precedenza

Chiamata anche *equità* (in inglese, *precedence*)

Indica particolari corrispondenze nelle sequenze temporali degli eventi

Proprietà di questo tipo vengono espresse da formule del tipo qPr oppure $p \rightarrow qPr$, dove P è l'operatore *precede* definito dall'equivalenza:

$$pPq \leftrightarrow_{def} \neg(\neg pUq).$$

Proprietà di questo tipo specificano il comportamento dei sistemi per l'allocazione delle risorse e per la fornitura di servizi, nei quali le richieste debbano essere soddisfatte nello stesso ordine in cui sono state formulate

Specifica, analisi, convalida e verifica di sistemi *time-* e *safety-critical* in logica temporale

- Due lezioni tenute il 29 Aprile prossimo dall'Ing. Angelo Morzenti, Politecnico di Milano

- Orario: 11-13 aula 4.2
  15-17 aula 2.3

- **Sommario**

  Uso di TRIO, una logica temporale con *metrica* sviluppata presso il Politecnico di Milano, nella specifica di alcuni esempi significativi di sistemi, particolarmente critici sotto l'aspetto della sicurezza e delle temporizzazioni. Si mostra come le specifiche in TRIO possano fornire un supporto efficace alle attività di *analisi* del sistema (come si comporta effettivamente il sistema descritto dalla specifica? quali sono le sue più interessanti proprietà?), di *convalida* della specifica (la specifica esprime in modo preciso, esauriente e non ambiguo i requisiti?) e di *verifica* della realizzazione (il sistema progettato soddisfa i requisiti, così come espressi dalla specifica?).

Specifiche algebriche

- Le strutture per la memorizzazione dati vengono viste come *tipi di dati astratti* (ADT), cioè insiemi di oggetti opachi
  - visibilità limitata
  - manipolazione solo attraverso un insieme di operazioni primitive ristretto e determinato
- Nell'ambito di un progetto dove un modulo realizza un ADT, si hanno due figure distinte, quella dei *clienti* del modulo che realizza il tipo di dato e quella dei suoi *realizzatori*
- Clienti, fanno uso delle funzionalità offerte da un tipo di dato
- Realizzatori, coloro che devono realizzare, attraverso la loro attività di programmazione, le astrazioni e le funzionalità previste per il dato, facendo uso direttamente di un linguaggio di programmazione o appoggiandosi sulle primitive di un tipo di dato già disponibile

- Descrizione, o *specifica*, di tipi di dati astratti

- Deve essere astratta, completa e formale

- Se la descrizione è astratta viene lasciata completa libertà al realizzatore nel modo in cui realizzare le funzionalità fornite da tipo di dato

- La completezza della specifica favorisce anche il cliente, che dispone di tutta l'informazione necessaria sulle caratteristiche del tipo di dato

- La formalità della descrizione permette di adottare metodi rigorosi per dimostrare la correttezza sia dell'implementazione di un tipo di dato, sia del suo uso per realizzare procedure più complesse

- Metodo algebrico, consiste di un insieme di notazioni e di metodi atti a fornire una specifica dei tipi di dati che abbia le caratteristiche sopra esposte

La specifica di un tipo di dato astratto

- Esempio di uno stack (pila)

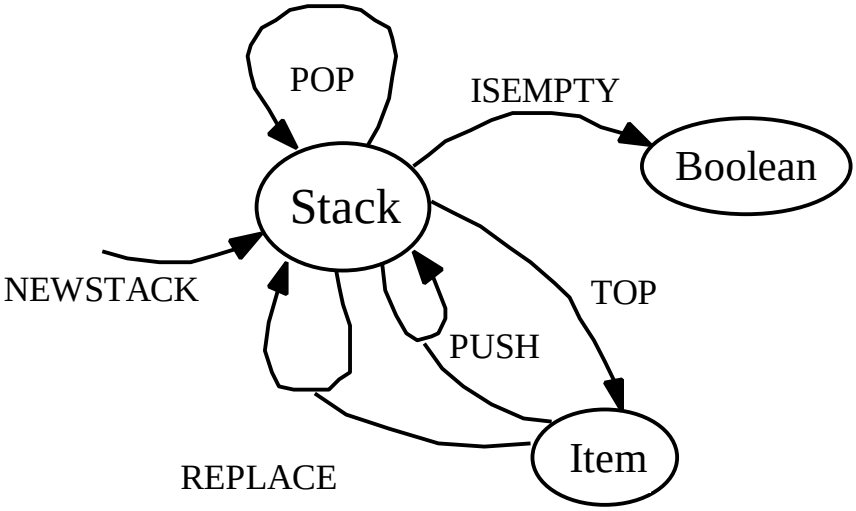
type Stack [Item]
uses Item, Boolean,
syntax
1 NEWSTACK : ∅ Stack,
2 PUSH : Item ∞ Stack ∅ Stack,
3 ISEMPTY : Stack ∅ Boolean,
4 POP : Stack ∅ Stack,
5 TOP : Stack ∅ Item,
6 REPLACE : Item ∞ Stack ∅ Stack,
semantics
for all stk **in** Stack; el, elm **in** Item;
7 ISEMPTY (NEWSTACK) = **true**,
8 ISEMPTY (PUSH (elm, stk)) = **false**,
9 POP (NEWSTACK) = **error**,
10 POP (PUSH (elm, stk)) = stk,
11 TOP (NEWSTACK) = **error**,
12 TOP (PUSH (elm, stk)) = elm,
13 REPLACE (elm, NEWSTACK) = **error**,
14 REPLACE (el, PUSH (elm, stk)) = PUSH (el, stk),
end Stack.

- La specifica è divisa in una parte *sintattica* e in una *semantica*

- La parte sintattica elenca le primitive del tipo di dato in via di definizione, rappresentate come funzioni di cui si fornisce il tipo, indicandone il dominio e il codominio

- I tipi utilizzati nella specifica della pila sono elencati nella seconda clausola, preceduti dalla parola chiave ***uses***

- Definisce una ***struttura algebrica eterogenea***



- *NEWSTACK* e *PUSH* operatori *costruttori* per il tipo di dato *Stack*

- *POP* e *REPLACE*, che effettuano ulteriori manipolazioni per produrre pile, sono detti *estensori*.

- *ISEMPTY* e *TOP*, *selettori.*

- La parte semantica è costituita da un insieme di *equazioni*, o assiomi, ossia da un insieme di uguaglianze tra termini

- I termini possono contenere variabili, le cui dichiarazioni sono riportate all'inizio della parte semantica

- Nell'esempio, *stk* rappresenta una pila, ed appartiene al tipo *Stack*, mentre *el* e *elm* sono componenti di pile, di tipo *Item*

- Anche i termini hanno un ***tipo***: per i termini costituiti da applicazioni di operatori il tipo del termine è il codominio della funzione applicata

- Le equazioni asseriscono delle ***uguaglianze*** tra ***termini*** dello stesso tipo

- Per convenzione, i nomi dei tipi iniziano con lettere maiuscole, quelli degli operatori sono completamente maiuscoli, e per le variabili sono completamente minuscoli

- Le parole chiave di questo elementare linguaggio di specifica sono scritte in minuscolo grassetto

- La specifica può descrivere, come nell'esempio, un tipo di dato astratto *parametrico* rispetto al tipo dell'elemento componente

- Identificatore *Item*, per indicare un generico tipo di dato cui gli elementi inseriti nella pila specificata appartengono

- Può essere *istanziato*, al momento di creare una particolare pila, a un particolare tipo

- Il concetto fondamentale del metodo algebrico per la specifica dei tipi di dati è che il tipo che si vuole specificare è completamente caratterizzato da tali ***equazioni***

- Le equazioni descrivono in modo formale e non ambiguo le ***proprietà rilevanti***, ma sono al contempo del tutto ***astratte***

- Non fanno riferimento alla struttura interna degli oggetti del tipo o al modo in cui le operazioni vengono realizzate

L'assioma 7 asserisce che la funzione *ISEMPTY*, applicata alla pila ottenuta dall'applicazione di *NEWSTACK*, dà valore *TRUE*, cioè che una pila appena creata è vuota e non contiene alcun elemento. L'assioma 8 stabilisce che ciò non è vero per una pila ottenuta dalla applicazione di *PUSH* a una pila preesistente, cioè afferma che una pila sulla quale l'ultima operazione effettuata è *PUSH* è sicuramente non vuota.
L'assioma 9 asserisce che il tentativo di togliere un elemento da una pila appena creata porta ad uno stato di errore; ciò è congruente con quanto appena affermato, ossia che una pila creata *ex novo* è vuota. L'assioma 10 dice che togliendo l'elemento posto in cima a una pila ottenuta aggiungendo un elemento a una pila data si ottiene la pila di partenza, cioè che l'elemento prelevato è proprio l'ultimo inserito, come si era detto nella descrizione informale della pila.
L'assioma 11 specifica che il tentativo di accedere all'elemento in testa ad una pila appena creata porta ad uno stato di errore, mentre l'assioma 12 asserisce che l'elemento visibile di una pila che deriva dall'inserzione di un elemento in una pila preesistente è proprio l'ultimo inserito, sempre coerentemente con la gestione LIFO caratteristica delle pile.
L'equazione 13 indica che il tentativo di sostituire l'elemento in cima ad una pila con un altro porta ad una situazione di errore, se la pila è appena stata creata. Secondo il successivo assioma 14, la stessa operazione, fatta su di una pila che deriva dall'inserzione di un elemento in un'altra pila, fornisce una pila equivalente a quella che si otterrebbe inserendo l'elemento, primo argomento di *REPLACE*, sulla pila di partenza.

- Le equazioni possono essere considerate come un insieme di assiomi di una teoria logica del primo ordine con l'uguaglianza come unico predicato

- I simboli di funzione sono definiti dalla parte sintattica della specifica

- Ad esempio l'assioma 14, che stabilisce una proprietà dell'operatore *REPLACE*, corrisponde alla seguente formula:

$$\forall stk \ \text{Stack} \ \forall el, elm \ \text{Item}$$

$$(\text{REPLACE}(el, \text{PUSH}(elm, stk)) = \text{PUSH}(el, stk) \ )$$

```
type Queue [Item]
   uses Item, Boolean,
   syntax
    1  NEWQ : ∅ Queue,
    2  ADDQ : Queue ∞ Item ∅ Queue,
    3  ISNEWQ : Queue ∅ Boolean,
    4  HEAD : Queue ∅ Item,
    5  DELETEQ : Queue ∅ Queue,
    6  APPENDQ : Queue ∞ Queue ∅ Queue,
   semantics
    for all  q, r in Queue;  i in Item;
    7  ISNEWQ (NEWQ) = true,
    8  ISNEWQ (ADDQ (q, i)) = false,
    9  HEAD (NEWQ) = error,
    10 HEAD (ADDQ (q, i)) = if ISNEWQ (q)
                            then i
                            else HEAD (q),
    11 DELETEQ (NEWQ) = NEWQ,
    12 DELETEQ (ADDQ (q, i)) = if ISNEWQ (q)
                            then NEWQ
                            else ADDQ (DELETEQ (q), i),
    13 APPENDQ (q, NEWQ) = q,
    14 APPENDQ (q, ADDQ (r, i)) =
                                ADDQ (APPENDQ (q, r), i),
end Queue.
```

- 7. *NEWQ* produce una coda vuota.
- 8. Una coda che contenga almeno un elemento non è considerata vuota.
- 9 e 10. Assiomi relativi a *HEAD*, la funzione che fornisce l'elemento in testa alla coda (quello che è stato inserito per primo). L'assioma 9 specifica che che da una ricerca del valore in testa a una coda vuota si ottiene un errore. L'assioma 10 indica che il risultato fornito dall'operazione *HEAD* applicata a una coda ottenuta dall'inserimento (*ADDQ*) nella coda *q* dell'elemento *i* coincide con *i* se *q* è la coda vuota, e quindi *HEAD* è stata applicata ad una coda con un solo elemento *i*, altrimenti viene restituito il valore in testa alla coda *q* (e cioè *HEAD(q)*). Un tale assioma può essere letto come la definizione ricorsiva di una procedura di calcolo.
- 11 e 12. Assiomi relativi a *DELETEQ*, che restituisce una coda uguale a quella ricevuta come parametro, privata del primo elemento, quello da più tempo in coda. L'assioma 11 dice che applicando *DELETEQ* a una coda vuota si ottiene la stessa coda vuota. L'assioma 12 specifica che il risultato ottenuto con l'applicazione di *DELETEQ* a una coda risultante dall'aggiunta di un elemento *i* ad una coda *q* è la stessa coda *q* se *q* è la coda vuota; altrimenti, se *q?NEWQ*, l'operatore *DELETEQ* restituisce la coda ottenuta con l'aggiunta di *i*, l'ultimo arrivato, in fondo alla coda restituita dalla applicazione di *DELETEQ* alla coda *q*.
- 13 e 14. Anche qui il meccanismo è ricorsivo: se si immagina la coda *r* come ottenuta con una serie di *n ADDQ* degli elementi $i_1 \ldots i_n$, appendere (cioè accodare ogni elemento di) *r* ad un'altra coda *q* equivale a fare la stessa sequenza di *n ADDQ* degli elementi $i_1 \ldots i_n$, nello stesso ordine in cui sono stati inseriti nella coda *r*, alla coda *q*.

- Il metodo algebrico permette non solo di caratterizzare le proprietà di un tipo di dato astratto nella sua specifica, ma anche di definire formalmente una sua possibile realizzazione usando un altro tipo, specificato anch'esso algebricamente

- Gerarchia di tipi a diverso livello di astrazione

- Ad esempio, coda mediante lista circolare:

```
implementation QueueByCirclist,
   representation QREP : Circlist [Item] ∅ Queue [Item],
   programs
    forall  c, c1: Circlist;  i: Item;
       NEWQ = QREP (CREATE),
       ADDQ (QREP (c), i) =
                        QREP (RIGHT (INSERT (i, c))),
       DELETEQ (QREP (c)) = QREP (DELETEC (c)),
       HEAD (QREP (c)) = VALUE (c),
       ISNEWQ (QREP (c)) = ISEMPTY (c),
       APPENDQ (QREP (c), QREP (c1)) =
                                    QREP (JOIN (c, c1)),
end QueueByCirclist.
```

✐ Esercizi su specifiche algebriche

1. Scrivere una specifica algebrica del tipo di dato Boolean, indicandone gli operatori costruttori ed etstensori

2. Scrivere una specifica del tipo *Set* in cui la duplicazione di elementi inseriti dal cliente più di una volta viene evitata

3. Scrivere una specifica del tipo di dato *Bag*, che è un insieme con elementi ripetuti

4. Specificare algebricamente il tipo di dato astratto *Pila con priorità*, di nome *PrStack*, in cui agli elementi impilati è associata una priorità. Un elemento inserito in pila viene posto *sotto* a tutti quelli già presenti in pila e aventi una priorità maggiore, e sopra a tutti quelli con priorità minore o uguale. La pila con priorità possiede gli operatori PR_TOP e PR_POP, aventi la seguente sintassi:

   PR_TOP: Priority ∞ PrStack ∅ Item,
   PR_POP: Priority ∞ PrStack ∅ PrStack,

   Tali operatori leggono e cancellano l'elemento, ultimo inserito in pila, avente priorità minore o uguale a una priorità data