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)

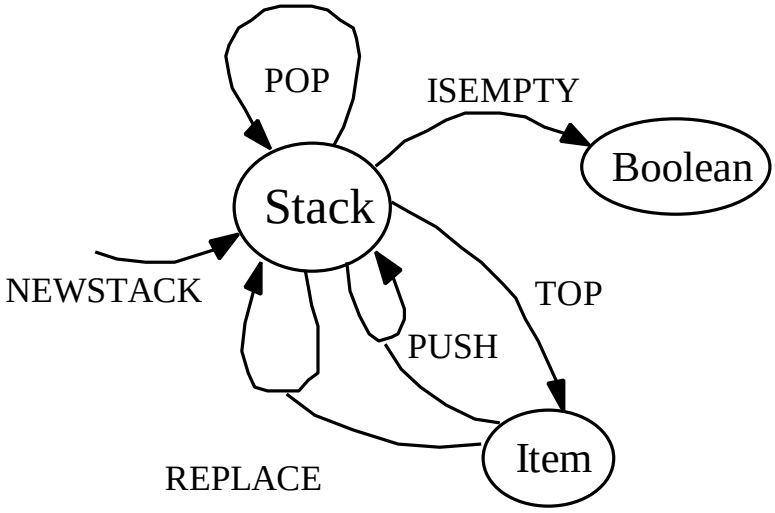
type Stack [Item]
uses Item, Boolean,
syntax
1 NEWSTACK : ∅ Stack,
2 PUSH : Item ∞ Stack ∅ Stack,
3 ISEMPTY : Stack ∅ Boolean,
4 POP : Stack ∅ Stack,
5 TOP : Stack ∅ Item,
6 REPLACE : Item ∞ Stack ∅ Stack,
semantics
for all stk **in** Stack; el, elm **in** Item;
7 ISEMPTY (NEWSTACK) = **true**,
8 ISEMPTY (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

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