Abstract Data Types: API, ADT

Sommario: 29 aprile, 2021

- Tipi Astratti: Struttura, API e ADT
- Un ADT per IntStack
- ADT: Modifichiamo l'implementazione
- Un API tante ADT
- ADT con polimorfismo: Definiamo il tipo 'a stack in OCaml
- ADT per valori non-modificabili:
  il tipo 'a stack in OCaml è per stack IMMUTABLE
- ADT per valori modificabili:
  il tipo stack<A> in Java è per stack polimorfi e MUTABLE
- Moduli
- Esercizi

Tipi Astratti: Principi

- **Tipo di Dato Astratto**: Una Collezione di Valori definita dall'insieme delle sole operazioni che possono essere usate per introdurli e operare con essi.

  - **Caratteristiche Distintive** sono le operazioni con cui sono completamente definiti e che si dividono in:
    - **Costruttori**: operazioni per introdurre tali valori (come Valori Calcolabili);
    - **Produttori**: operazioni che calcolano tali valori (come Valori Calcolabili);
    - **Modificatori**: operazioni che modificano (componenti di) tali valori (solo per Tipi Astratti modificabili e strutturati);
    - **Osservatori**: operazioni che accedono ai componenti di tali valori (solo per Tipi Astratti strutturati);

Tipi Astratti: Principi/2

- **Tipo di Dato Astratto**: Una Collezione di Valori definita dall'insieme delle sole operazioni che possono essere usate per introdurli e operare con essi.
  - **Caratteristiche Distintive** sono le operazioni con cui sono completamente definiti e che si dividono in:...
  - **Rappresentazione** di tali valori **non è visibile**, e **nemmeno utile** per programmare con tali valori
  - **Implementazione** delle operazioni **non è visibile**[1], e **nemmeno utile** per programmare con tali valori
  - **Rappresentazione e Implementazione** possono essere cambiate senza che il comportamento del programma cambi
  - **Unità di Programmazione per Dati** della Programmazione Strutturata: Introducono nuove collezioni di valori **nascondendo dettagli implementativi che restano però localizzati** nella definizione del tipo astratto.

[1] all'esterno della definizione

Tipi Astratti: API & ADT

- **Tipo di Dato Astratto**: Una Collezione di Valori definita dall'insieme delle sole operazioni che possono essere usate per introdurli e operare con essi.

  - Un tipo astratto è una definizione di tipo con la seguente struttura generale:
    - typeName: che fornisce il nome del nuovo tipo
    - signature: che contiene la segnatura di tutte le operazioni visibili e quindi utilizzabili per il nuovo tipo
    - implementazione: che contiene l'intera implementazione (rappresentazione e implementazione op.)

  - In una sintassi concreta ispirata a Standard ML:

```
abstype typeName{
      private section:  Definizioni di tipo
                per la rappresentazione
   signature
      public section:  Nomi e tipi delle operazioni
              utilizzabili all'esterno
   operations
      private section:  Definizioni delle operazioni,
              incluse le ausiliarie
}
```

  - **A**pplicative **P**rogramming **I**nterface = typeName + signature

Un ADT IntStact: Stacks bounded di int

```
abstype Int_Stack{
    type Int_Stack = struct{
                        int p[100];
                        int n;
                        int top;
    };
signature
    Int_Stack create_stack();
    void push(Int_Stack s, int k);
    int top(Int_Stack s);
    void pop(Int_Stack s);
    bool empty(Int_Stack s);

operations
    Int_Stack create_stack(){
        int_stack s = new Int_Stack();
        s.n = 0;
        s.top = 0;
        return s;
    }
    void push(Int_Stack s, int k){
        if (s.n == 100) error;
        s.n = s.n+1;
        s.p[s.top] = k;
        s.top = s.top+1;
    }
    int top(Int_Stack s){
        return s.p[s.top];
    }
    void pop(Int_Stack s){
        if (s.n == 0) error;
        s.n = s.n-1;
        s.top = s.top-1;
    }
    bool empty(Int_Stack s){
        return (s.n == 0);
    }
}
```

Un rifrasamento in C: Stacks bounded di int

```
//abstype IntStack{
    struct stackStruct{
        int p[100];
        int n;
        int top;
    };
    typedef struct stackStruct stackElem;
    typedef stackElem* IntStack;
/*
    signature
        IntStack createStack();
        void push(IntStack s, int k);
        int top(IntStack s);
        void pop(IntStack s);
        bool empty(IntStack s);

    operations
 */
    IntStack createStack(){
        IntStack s = (IntStack) malloc(sizeof(stackElem));
        s->n = 0;
        s->top = 0;
        return s;
    }
    void push(IntStack s, int k){
        //if (s->n == 100) error;
        s->n = s->n+1;
        s->p[s->top] = k;
        s->top = s->top+1;
    }
    int top(IntStack s){
        return s->p[s->top];
    }
    void pop(IntStack s){
        //if (s->n == 0) error;
        s->n = s->n-1;
        s->top = s->top-1;
    }
    bool empty(IntStack s){
        return (s->n == 0);
    }
}
```

Accesso della sola signature (interfaccia)

Tipo Astratto: Rappresentazione dei valori e implementazione delle operazioni sono accessibili solo all'interno della definizione.

- Si consideri il seguente frammento di programma di un linguaggio a blocchi, scope statico, ADT e struttura C-like di comandi ed espressioni. Il frammento utilizza lo ADT `Int_Stack` e contiene alcuni errori. Quali e Perchè?

```
abstype Int_Stack{...};
int k = 0;
Int_Stack stack1, stack2;
stack1 = create_stack();
stack1 = push(stack1, 1);
stack1 = push(stack1, 5);
stack2 = stack1;
if (stack1.n>0) stack1.p[top − 1] = stack1.p[top];
while (!empty(stack2)){
                k = k + top(stack2);
                pop(stack2);
                stack2.top = k;
                }
```

Valore Astratto, Presentazione e Stato Concreto

- **Valori Astratti**. Insieme **A** dei valori esprimibili con un ADT.

- **Presentazione.** Le forme sintattiche con cui possono essere mostrati i valori di **A**

Esempio: I valori di $\mathtt{A_{intStack}}$, hanno la seguente presentazione:

> $\mathtt{A_{intStack}}$ è insieme di sequenze modificabili della forma $[\mathtt{x_1}, \ldots, \mathtt{x_k}]$, quando $\mathtt{k} > 0$, [], altrimenti.
> L'intero $\mathtt{x_k}$ è l'ultimo inserito ed è il primo che può essere acceduto.

- **Stati Concreti.** Insieme **C** degli stati definibili nell'ADT per rappresentare **A**.

Esempio: $\mathtt{C_{intStack}}$ è insieme di triple definite da:

```
type Int_Stack = struct{
        Int P[100];
        Int n;
        Int top;
}
```

Relazione tra Valore Astratto e Stato Concreto: AF e I

- **Valori Astratti**. Insieme **A** dei valori esprimibili con un ADT.
  $A_{\texttt{intStack}}$ è insieme di sequenze modificabili della forma $[\texttt{x}_1, \ldots, \texttt{x}_\texttt{k}]$, quando $\texttt{k} > 0$, [], altrimenti.
  L'intero $\texttt{x}_\texttt{k}$ è l'ultimo inserito ed è il primo che può essere acceduto.

- **Stati Concreti.** Insieme **C** degli stati definibili nell'ADT per rappresentare **A**.

```
type Int_Stack = struct{
        Int P[100];
        Int n;
        Int top;
}
```

Funzione di Astrazione. È una funzione **AF** : **C** $\rightarrow$ **A** che associa ad ogni **stato concreto legale** la presentazione del valore astratto rappresentato

Esempio
$\texttt{AF(c)} = [] \texttt{ if c.n} == 0$
$\texttt{AF(c)} = [\texttt{x}_1, \ldots, \texttt{x}_\texttt{k}] \texttt{ con } \texttt{x}_\texttt{i} == \texttt{c.P[i} - 1](\forall \texttt{i} \in [1..\texttt{k}]), \texttt{ if c.n} == \texttt{k} > 0$

Invariante di Rappresentazione. È un predicato che definisce gli **stati concreti legali** di **C**

Esempio
$\texttt{I(c)} = (\texttt{c.n} >= 0)\&(\texttt{c.n} == \texttt{c.top})\&(\texttt{c.n} <= 100)$

ADT Polimorfo in OCaml: Stack Immutable


```
exception Error ;;

module type STACK =
 sig type 'a stack
  val create_stack: unit -> 'a stack
  val push: 'a stack -> 'a -> 'a stack
  val top: 'a stack -> 'a
  val pop: 'a stack -> 'a stack
end;;


module Stack =
 (struct
  type 'a sk = E | SK of 'a sk * 'a
  type 'a stack = M of int * ('a sk)
  let create_stack = fun () -> M(0,E)
  let push (M(n,sk)) x =
        if n=100 then raise(Error)
        else M(n+1,SK(sk,x))
  let top (M(n,sk)) = match sk with
                E -> raise Error
                | SK(sk,v) -> v
  let pop (M(n,sk)) = match sk with
                E -> raise Error
                | SK(sk,v) -> M(n-1,sk)
end:STACK);;
```

ADT Polimorfo in Java: Stack Mutable

StackADT.java 4/23/17 7:20 PM

```
abstract class STACK <A>{ //A Java API for the type Stack
    public abstract void create_Stack();
    public abstract void push (A x);
    public abstract A top (A x);
    public abstract void pop();
}

class Stack <A> extends STACK <A>{
            //MUTABLE ABSTRACT VALUES
    private Vector <A> P;
    private int n;
    private int top;

   public void create_Stack(){
       P = new Vector <A>();
       n = 0;
       top = -1;
   }
   public void push(A x){
       if (n == 100) throw new Error();
       n = n+1;
       top = top + 1;
       P.add(top,x);
   }
   public A top(A x){
       if (n == 0) throw new Error();
       return (P.get(top));
   }
   public void pop(){
       if (n == 0) throw new Error();
       n = n-1;
       top = top - 1;
   }
}
```

ADT: Osservazioni

- L'ADT per stack in linguaggio C-like implementa correttamente le funzioni `AF` e `I`, date?
  Cosa significa "correttamente"?
  Quali proprietà devono essere verificate per la correttezza?

Proprietà dell'Invariante Il codice di ogni Costruttore e di ogni Modificatore deve modificare uno stato concreto `c` in uno `c'` tale che se `I(c)` allora `I(c')`.

Proprietà dei Produttori Il codice di ogni Produttore non deve modificare lo stato concreto.

ADT: Osservazioni - 2

- Come sono definite le funzioni AF e I per gli ADT dati in Ocaml e in Java, rispettivamente?

- L'ADT per stack in Ocaml è immutable perchè, oltre a costruttori e osservatori, ha solo produttori.
  L'ADT per stack in Java è mutable perchè, oltre a costruttori e osservatori, ha solo modificatori.
  L'ADT per stack nel linguaggio C-like di che tipo è?

- L'operazione push dell'ADT per stack nel linguaggio C-like è un produttore o un modificatore?

Moduli

- ADT = Unità di Programmazione per Programmazione in Piccolo
  - operano su un tipo di dato per localizzare la definizione e
  - limitare la visibilità in accordo alla regola della "sola segnatura"

- Moduli = Unità di Programmazione per Programmazione in Grande
  - Operano su partizioni di sistemi di grandi dimensioni
    - Trattandole come parti autonome e
    - Compilabili in modo indipendente e
    - Con caratteristiche di modificabilità simili agli ADT
  - Raggruppano più dichiarazioni (dati e/o funzioni), e
  - Definiscono regole di visibilità per tali dichiarazioni
    - Stabilendo ciò che è pubblico e ciò che non lo è
    - Importando da moduli ed esportando solo su altri

Modulo: Costrutti Imports, Public, Private

```
module Buffer imports Counter{
   public
      type Buf;
      void insert(reference Buf f, int n);
      int get(Buf b);
      Count c; // how many times buffer has been used
   private imports Queue{
        type Buf = Queue;
      void insert(reference Buf b, int n){
         inqueue(b,n);
         inc(c);
      }
      int get(Buf b){
         return dequeue(b);
         inc(c);
      }
      init_counter(c);    // module initialisation part
}
module Counter{
   public
      type Count;
      void init_counter(reference Count c);
      int get(Count c);
      void inc(reference Count c);
   private
      type Count = int;
      void init_counter(reference Count c){
         c=0;
      }
      int get(Count c){
         return c;
      }
      void inc(reference Count c){
         c = c+1;
      }
}
```

Esercizi

1. Quali tra le seguenti proprietà di un Tipo Astratto non sono presenti nei Tipi Algebrici? Costruttori, Modificatori, Produttori, Non Visibilità della Rappresentazione, Non Visibilità dell'Implementazione delle Operazioni, Unità di Programmazione per Dati.
2. Differenze tra API e ADT in un Tipo Astratto.
3. Perchè la definizione di `Int_Stack`, nella slide seguente, è errata? Indicare gli errori e le cause.
4. Perchè il rifrasamento in C, incluso nelle slide, non definisce un Tipo Astratto in C? Quali proprietà sono violate?
5. Quali errori sono contenuti nel frammento di codice C-like, incluso nelle slide, che introduce ed usa il tipo `Int_Stack`? Per ognuno si indichi posizione, causa e possibile correzione.
6. Si consideri l'ADT polimorfo stack per valori stack Immutable scritto in OCaml ed incluso nelle slides. Si indichino:
   (a) valori astratti A,
   (b) stati concreti C,
   (c) funzione di astrazione AF,
   (d) invariante di rappresentazione I.
7. (a) Si dica perchè lo ADT polimorfo scritto in OCaml è per valori Immutable.
   (b) Si dica perchè lo ADT polimorfo scritto in Java è per valori Mutable.

Altri esercizi in EserciziL10.pdf

Una ADT IntStack da Evitare

```
abstype Int_Stack{
   type Int_Stack = struct{
                        int P[100];
                        int n;
                        int top;
   }
   signature
      Int_Stack create_stack();
      Int_Stack push(Int_Stack s, int k);
      int top(Int_Stack s);
      Int_Stack pop(Int_Stack s);
      bool empty(Int_Stack s);
   operations
      Int_Stack create_stack(){
         Int_Stack s = new Int_Stack();
         s.n = 0;
         s.top = 0;
         return s;
      }
      Int_Stack push(Int_Stack s, int k){
         if (s.n == 100) error;
         s.n = s.n + 1;
         s.P[s.top] = k;
         s.top = s.top + 1;
         return s;
      }
      int top(Int_Stack s){
         return s.P[s.top];
      }
      Int_Stack pop(Int_Stack s){
         if (s.n == 0) error;
         s.n = s.n - 1;
         s.top = s.top - 1;
         return s;
      }
      bool empty(Int_Stack s){
         return (s.n == 0);
      }
}
```

Si risponda:
(1) Un produttore può essere anche modificatore?
(2) Sia sk uno stack non vuoto. Quali stack calcolano le seguenti espressioni:
push(pop(sk),top(sk)) ?
pop(push(sk,n)) ?
(3) In cosa differisce lo stack sk iniziale da quelli calcolati dalle due espressioni, sopra?

Un esercizio in OCaml

- Come sono definite le funzioni AF e I per l'ADT Ocaml di Stack?

- **Soluzione**

  - Si parte sempre da una presentazione dei valori.
    ['a stack] è insieme di sequenze immutable della forma $[x_1, \ldots, x_k]$, quando $k > 0$, [], altrimenti.
    Gli elementi $x_i \in$ ['a] e x_k è l'ultimo inserito ed è il primo che può essere acceduto o estratto.

  - Si considera l'insieme degli stati concreti introdotti dalla rappresentazione.
    ```
    type 'a sk = E | SK of 'a sk * 'a
    type 'a stack = M of int * ('a sk)
    ```

  - Si definisce la funzione di astrazione sul sottoinsieme dei legali.
    $AF(c) = []$ if $c = M(0, \_)$
    $AF(c) = [x_1, \ldots, x_{k-1}, x_k]$ if $(c = M(k, v))\&(k > 0)\&(v = SK(w, x_k))\&(w = SK(\ldots(SK(x_1, E), \ldots), x_{k-1})$

  - Si definisce l'invariante per il sottoinsieme dei legali.
    $I(c) = (c = M(k, v)\&(k =$ size $v)$
    per size di seguito definita:
    ```
    size = function E -> 0 | SK(w,_) -> 1 + size w
    ```