Lezione 26-27-28
Tipi Astratti e Astrazione di Dato

prof. Marco Bellia, Dip. Informatica, Università di Pisa

May 2-7, 2012

Tipi Astratti e Astrazione di Dato

- Tipi e Dati Concreti: Definizione, Allocazione e Accesso
- Tipi Astratti e Astrazione di dato: API e ADT
- Astrazione di dato: Domini Sintattici
- Astrazione di dato: Domini Semantici e Funzioni Semantiche
- Tipi Astratti e Astrazione di dato: Implementazione
- Tipi Astratti e Astrazione di dato: Uso e Applicazioni

Types and Data Abstraction: Semantics of ADT/5

Example

Apply it to the ADT `P`, we introduced for cartesian ponts, and to the code below:

```
P x;
x = mk(3,5); ...  fst(x)...
```

$$\mathcal{D}[\![\texttt{P x}]\!]_\rho(\texttt{s}) = (\rho_2, \texttt{s}_2)$$

where:

$\rho_2(\texttt{x}) = \texttt{l}_\texttt{x} \oplus \rho_2(\texttt{mk}, \texttt{fst}, \texttt{snd}) = \sigma(\texttt{mk}, \texttt{fst}, \texttt{snd}) \oplus \rho$

$\texttt{s}_2 = \texttt{upd}(\texttt{l}_\texttt{x}, \texttt{up}(\sigma, \bot_{\texttt{Mem}}), \texttt{s}_1)$

where: $\texttt{s}_1 = \texttt{allocate}(\texttt{ADT}, \texttt{s})$

σ is the access key of ADT P.

$$\mathcal{E}[\![\texttt{Callmk(byValue 3, byValue 5)}]\!]_{\rho_2}(\texttt{s}_2) = (\texttt{v}, \texttt{s}_3)$$

where:

v is a value of form λkey.e, for suitable e

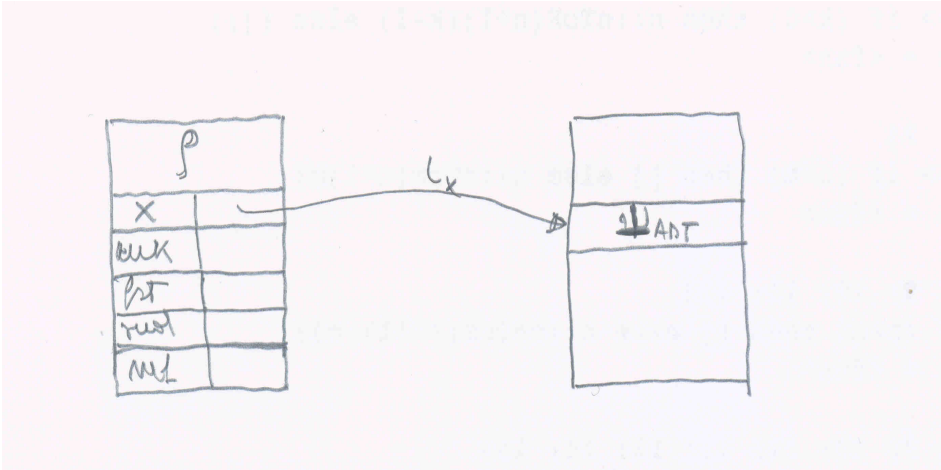
$\texttt{s}_3 = \texttt{g}(3,5)(\texttt{s}_2)$, where $\texttt{F}(\texttt{g}) = \rho_2(\texttt{mk})$.

...

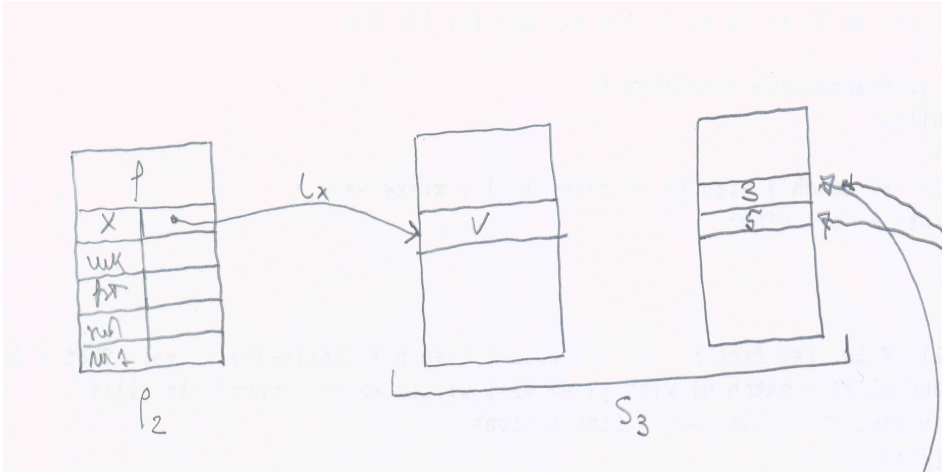
to be completed...

Types and Data Abstraction: Semantics of ADT/6

- A graphical view of the solution:



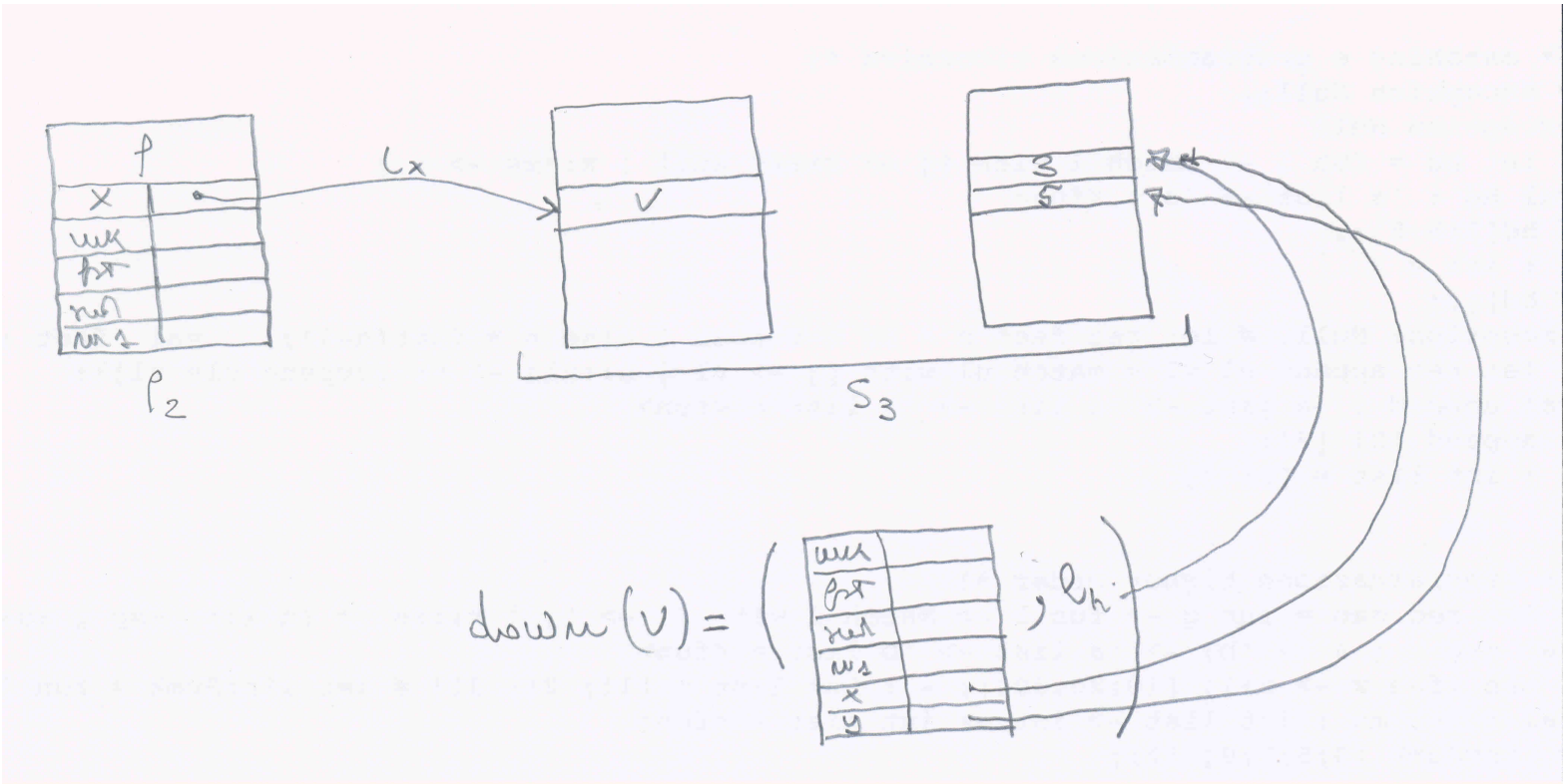
- `P x;`



- `x = mk(3,5);`

Types and Data Abstraction: Semantics of ADT/7

- A graphical view of the solution:
- `x = mk(3,5);`: A complete view



Tipi e Dati Astratti: API

Un API in Ocaml per un ADT per RELAZIONI (binarie, polimorfe)

Example

```
module type RELAZIONE =
 sig type ('a,'b) relazione
  val relazioneC: unit -> ('a,'b) relazione
  val isUno: ('a,'b) relazione -> 'a -> bool
  val isDue: ('a,'b) relazione -> 'a -> bool
  val getUno: ('a,'b) relazione -> 'b -> 'a list
  val getDue: ('a,'b) relazione -> 'a -> 'b list
end;;
```

- Si introduce un tipo astratto con nome ('a,'b)`relazione` (polimorfo nelle variabili di tipo a e b).
- Si utilizza un modulo con nome RELAZIONE e "ruolo" `type`.
- Il modulo contiene nell'ordine, la segnatura del tipo e delle operazioni visibili dall'esterno.

Tipi e Dati Astratti: ADT/1

Un ADT in Ocaml per RELAZIONI (binarie, polimorfe)

Example

```
module Relazione =
 (struct
  type ('a,'b) relazione = ('a * 'b) list
  let relazioneC = []
  let isUno(r,x) = List.fold_right (||) (List.map(fun u -> fst(u)=x) r) false
  let isDue(r,y) = List.fold_right (||) (List.map(fun u -> snd(u)=x) r) false
  let getUno (r,y) = let g = fun u ->if(snd(u)=y)then[fst(u)]else[]
                     in List.fold_right(List.append)(List.map g r) []
  let getDue (r,x) = let g = fun u ->if(fst(u)=x)then[snd(u)]else[]
                     in List.fold_right(List.append)(List.map g r) []
end:RELAZIONE);;
```

- Si fornisce un'implementazione, ADT, del tipo astratto con nome `('a,'b)relazione`.
- Si utilizza un modulo con nome RELAZIONE e "ruolo" ordinario e contenente uno `struct...end`.
- Il modulo contiene nell'ordine, la definizione del tipo astratto (e di altri tipi ausiliari), delle operazioni visibili dall'esterno e di quelle

Tipi e Dati Astratti: ADT/2

Example

```
module Relazione =
 (struct
  type ('a,'b) relazione = ('a * 'b) list
  let relazioneC = []
  let isUno(r,x) = List.fold_right (||) (List.map(fun u -> fst(u)=x) r) false
  let isDue(r,y) = List.fold_right (||) (List.map(fun u -> snd(u)=x) r) false
  let getUno (r,y) = let g = fun u ->if(snd(u)=y)then[fst(u)]else[]
                     in List.fold_right(List.append)(List.map g r) []
  let getDue (r,x) = let g = fun u ->if(fst(u)=x)then[snd(u)]else[]
                     in List.fold_right(List.append)(List.map g r) []
end:RELAZIONE);;
```

- Il modulo contiene nell'ordine, la definizione del tipo astratto (e di altri tipi ausiliari), delle operazioni visibili dall'esterno e di quelle ausiliarie.
- Tutte le definizioni ausiliarie non sono visibili dall'esterno
- `struct...end:A` qualifica ADT come relativo al mod. RELAZIONE
- Un API può avere più ADT differenti che forniscono differenti implementazioni

Tipi e Dati Astratti: Implementazione

- Un repository contiene tutte le definizioni degli ADT accessibili attraverso una chiave unica per ogni ADT
- Repository, chiavi e funzioni up e down, per ogni chiave, e loro inserimento nella sintassi astratta possono essere generati dal compilatore.

Fondamenti di Programmazione Funzionale

prof. Marco Bellia, Dip. Informatica, Università di Pisa

May 7, 2012

Programmazione Funzionale

- Syntax Essentials
- Language Domains and Semantic Functions
- Operational Semantics: Term Reduction
- Astrazione di dato: Domini Semantici e Funzioni Semantiche
- Remark: Functional v. Logical variables
- Remark: Matching v. Unification
- Remark: Currying and Higher Order
- Programming Methodologies: Recursive, Higher Order, Tail Recursive, Iterative and Combinators based

Linguaggi Funzionali: Essenziali di Sintassi

Table20 – Linguaggi Fnzionali : Sintassi

Struttura dei programmi

D ::= F | T | ...	
F ::= I= A	
A ::= funI $\rightarrow$ E	
E ::= A | E E | I | ...	
T ::= ...	(*scalari*, *tuple freccia*, ...)
| ...	(*liste* : *fond*. *metodol*.)
| ...	(*Concreti* : *fond*. *metodol*.)
| ...	(*Astratti* : *fond*. *metodol*.)

Linguaggi Funzionali: Domini Sintattici e Semantici/1

- Li abbiamo già visti nello studio dei meccanismi dei linguaggi
- Sintattici: Dichiarazioni, Espressioni (niente comandi) + ausiliari (tipi, ..)
- Semantici: Ambienti, Memoria (solo per allocazione dinamica dei valori)
- Funzioni Semantiche: le solite per dichiarazioni ed espressioni

- La Trasparenza Referenziale conduce ad una particolare Semantica Operazionale: Riduzione di Termine

Linguaggi Funzionali: Semantica a Riduzione (di Termine)

- $\alpha -$ *riduzione*

$$\texttt{fun } \texttt{I}_1 \rightarrow \texttt{E} \Longrightarrow \texttt{fun } \texttt{I}_2 \rightarrow [\texttt{I}_2/\texttt{I}_1]\texttt{E} \quad \text{when } \texttt{I}_2 \notin \texttt{Free(E)}$$

- $\beta -$ *riduzione*

$$(\texttt{fun } \texttt{I} \rightarrow \texttt{E})\texttt{E}_0 \Longrightarrow [\texttt{E}_0/\texttt{I}]\texttt{E} \quad \text{when } \texttt{Free}(\texttt{E}_0) \cup \texttt{Bound(E)} = \{\}$$

- altre regole per eventuali altri costrutti

Remark: Variabili Funzionali vs Variabili Logiche

- Possiamo scrivere append nei due linguaggi

```
app [] y = y                    app([], Y, []).
app(e::es) y =                  app([E|Es], Y, [E|Z]):-
  e::(app es y)                   app(Es,Y,Z).
```

- le variabili sono tutte quantificate universalmente.
- Ma riusciamo a cogliere bene la differenza se "applichiamo"

```
app v1 v2                       app(t1(X),t2(X),t3(X)).
```

- Infatti `v1 v2` sono valori (termini ground, in LP), mentre $\overline{X}$ sono variabili logiche, ovvero quantificate esistenzialmente, che possono occorrere nei termini `t1(`$\overline{X}$`),t2(`$\overline{X}$`),t3(`$\overline{X}$.

Remarks:

- Matching vs Unification
  - il matching coinvolge sempre un termine ground (valore) e un termine pattern (su termini ground)
  - l'unificazione coinvolge un termine che può contenere variabili logiche e un termine pattern (su termini con variabili logiche).
- Currying e Higher Order
  - Sia $(+)$ il currying di $+$, allora succ $= (+)1$ è un valore, una funzione.

Programming Methodologies:

- Recursive
- Higher Order
- Tail Recursive
- Iterative and Combinators based
- Esempi di ciascuna: file OcamlView