Lezione 26
Tipi Astratti e Astrazione di Dato

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

May 2, 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
- Esercizi

Tipi e Dati Concreti: Definizione, Allocazione e Accesso

Ricordiamo la struttura generale (e applichiamola ad un record)

Table15 – Tipi e Dati Concreti

Domini Sintattici

$\mathtt{D} ::= \ldots \mid \mathtt{T\ I}$ (*naming : Allocazione Statica*)
$\mid \mathtt{type\ I} = \mathtt{T}$ (*Definizione di Tipo*)
$\mid \ldots$

$\mathtt{E} ::= \ldots \mid \mathtt{Talloc\ (T)}$ (*Allocazione Dinamica*)
$\mid \mathtt{Den(E.I_k)} \mid \mathtt{E.I_k} \mid \mathtt{Val(I)}$ (*Accesso*)
$\mid \ldots$

Domini Sintattici Ausiliari

$\mathtt{T} ::= \mathtt{T_A} \mid \ldots \mid \mathtt{T_F} \mid \mathtt{I} \mid \ldots$ (*anche polimorfi*)
$\mathtt{T_A} ::= \mathtt{int} \mid \ldots \mid \mathtt{bool}$ (*atomici*)
$\mathtt{T_D} ::= \mathtt{VL_{inf}} \,..\, \mathtt{VL_{sup}} \mid \ldots \mid *\mathtt{T}$ (*derivati*)
$\mathtt{T_S} ::= \mathtt{Rec\ I_1{:}T_1 \ldots I_n{:}T_n\ End} \mid \ldots$ (*strutturati*)
$\mathtt{T_F} ::= \mathtt{T_1} \times \ldots \times \mathtt{T_n} \rightarrow \mathtt{T} \mid \ldots$ (*funzione*)

Tipi e Dati Concreti: Definizione, Allocazione e Accesso

La sintassi concreta nasconde meccanismi semplificando la scrittura di programmi. L'analisi statica deve provvedere a mostrare tali meccanismi. Ad esempio

Example

Si mostri dove la sintassi astratta usa `Val` e `Den`, nel seguente frammento di programma C.

```
typedef struct {char* Title, Auth, ISBN; float Price;
 }book;
book [100] line;
book b1, b2;
b1={"Un Libro","Art, Smith","00 32 X",,}
b1.Price = b2.Price = 70.00 + 5.75;
...
line[x].Title = (line[x].ISBN==b2[ISBN]) ? b2.Title : line[y]Title;
```

Dichiarazione e Allocazione Statica di Valori Record/1

Dichiarazione di un binding per I: L'identificatore è legato ad una variabile record allocata staticamente (al momento del caricamento del programma, ad esempio)

Table15.1 – variabile Record

Funzioni Semantiche

$$\mathcal{D}[\![\mathtt{D}]\!]_\rho : \mathtt{Store} \to (\mathtt{Env} \times \mathtt{Store})_\perp$$

$$\mathcal{D}[\![\mathtt{Rec\ I_1{:}T_1 ... I_n{:}T_n\ End\ I}]\!]_\rho(\mathtt{s}) = \mathtt{Let}\{\mathtt{s_0} = \mathtt{s}, \forall \mathtt{T_i}\,.(\mathtt{l_i}, \mathtt{s_i}) = \mathit{allocate}(\mathtt{T_i}, \mathtt{s_{i-1}})\} \{\forall \mathtt{I_i}\,.\gamma(\mathtt{I_i}) = \mathtt{l_i}\} (\mathtt{bind}(\mathtt{I}, \mathtt{E_D}(\gamma), \rho), \mathtt{s_n})$$

$$\mathcal{E}[\![\mathtt{E}]\!]_\rho : \mathtt{Env} \to \mathtt{Store} \to (\mathtt{Env} \times \mathtt{Store})_\perp$$

$$\mathcal{E}[\![\mathtt{Den(E.I_k)}]\!]_\rho(\mathtt{s}) = \mathtt{Let}\{\mathtt{E_D}(\gamma) = \mathcal{E}[\![\mathtt{E}]\!]_\rho(\mathtt{s})\}(\gamma(\mathtt{I_k}), \mathtt{s})$$

$$\mathcal{E}[\![\mathtt{E.I_k}]\!]_\rho = \ldots$$

Funzioni Ausiliarie e Domini Ausiliari

$$\mathtt{E_D} : \mathtt{EnvL} \to \mathtt{Den}$$

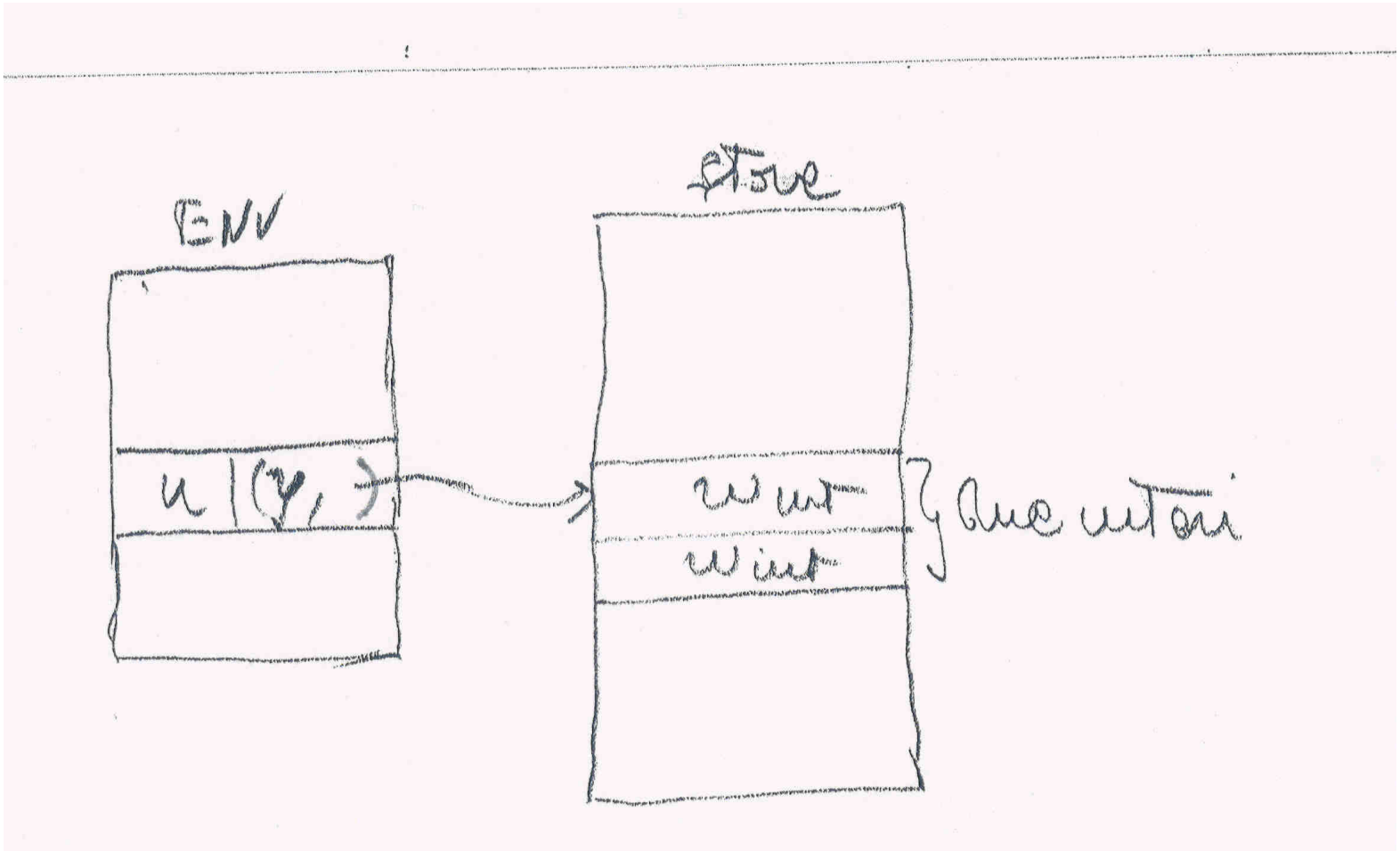
$$\mathtt{Den} ::= \ldots + \mathtt{EnvL}$$

$$\mathtt{EnvL} ::= \mathtt{I} \to \mathtt{Den}$$

Dichiarazione e Allocazione Statica di Valori Record/2

Implementazione:

- $\mathtt{E_D}(\gamma)$ è implementabile mediante una coppia contenente γ e la prima locazione del blocco allocato per il record
- graficamente: `Rec int x, int y End u`.



Allocazione Dinamica di Valore Record

Creazione di un valore modificabile (variabile) di tipo record

Table15.1.1 – Puntatori Record

Funzioni Semantiche

$$\mathcal{E}[\![\mathtt{E}]\!]_\rho : \mathtt{Store} \to (\mathtt{Val} \times \mathtt{Store})_\perp$$

$$\mathcal{E}[\![\mathtt{Talloc(Rec\ I_1 : T_1 ... I_n : T_n\ End)}]\!]_\rho(\mathtt{s}) =$$
$$\mathtt{Let}\{\mathtt{s_0} = \mathtt{s}, \forall \mathtt{T_i}.(\mathtt{l_i}, \mathtt{s_i}) = \mathit{allocate}(\mathtt{T_i}, \mathtt{s_{i-1}})\}$$
$$\{\forall \mathtt{I_i}.\gamma(\mathtt{I_i}) = \mathtt{l_i}\}$$
$$(\mathtt{E_V}(\gamma), \mathtt{s_n})$$

Funzioni Ausiliarie e Domini Ausiliari

$\mathtt{E_V} : \mathtt{EnvL} \to \mathtt{Val}$ $\qquad$ $\mathtt{E_M} : \mathtt{EnvL} \to \mathtt{Mem}$

$\mathtt{Val} ::= \ldots + \mathtt{EnvL}$ $\qquad$ $\mathtt{Mem} ::= \ldots + \mathtt{EnvL}$

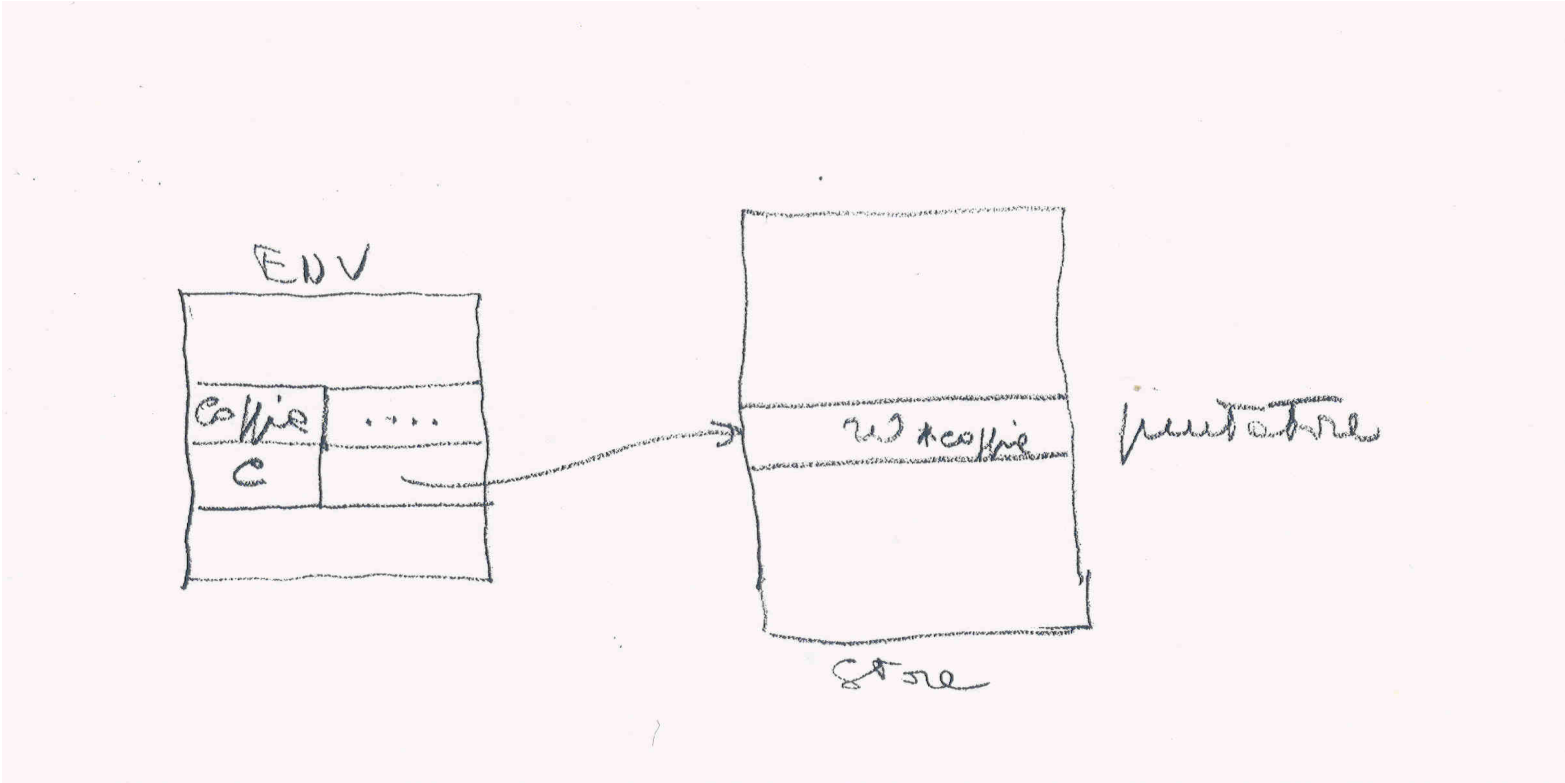
$\mathtt{EnvL} ::= \mathtt{I} \to \mathtt{Den}$

Implementazione: `Store` contiene anche una memoria Heap, H, ed *allocate* alloca su H sia i componenti sia la struttura di accesso che implementa la funzione γ.

Dichiarazione e Allocazione Statica di Valori Record/2

Implementazione:

- $\mathtt{E_D}(\gamma)$ è implementabile mediante una coppia contenente γ e la prima locazione del blocco allocato per il record
- graficamente: $\mathtt{type\ Coppia} = \mathtt{Rec\ int\ x, int\ y\ End}; \mathtt{Coppia} * \mathtt{c};$.

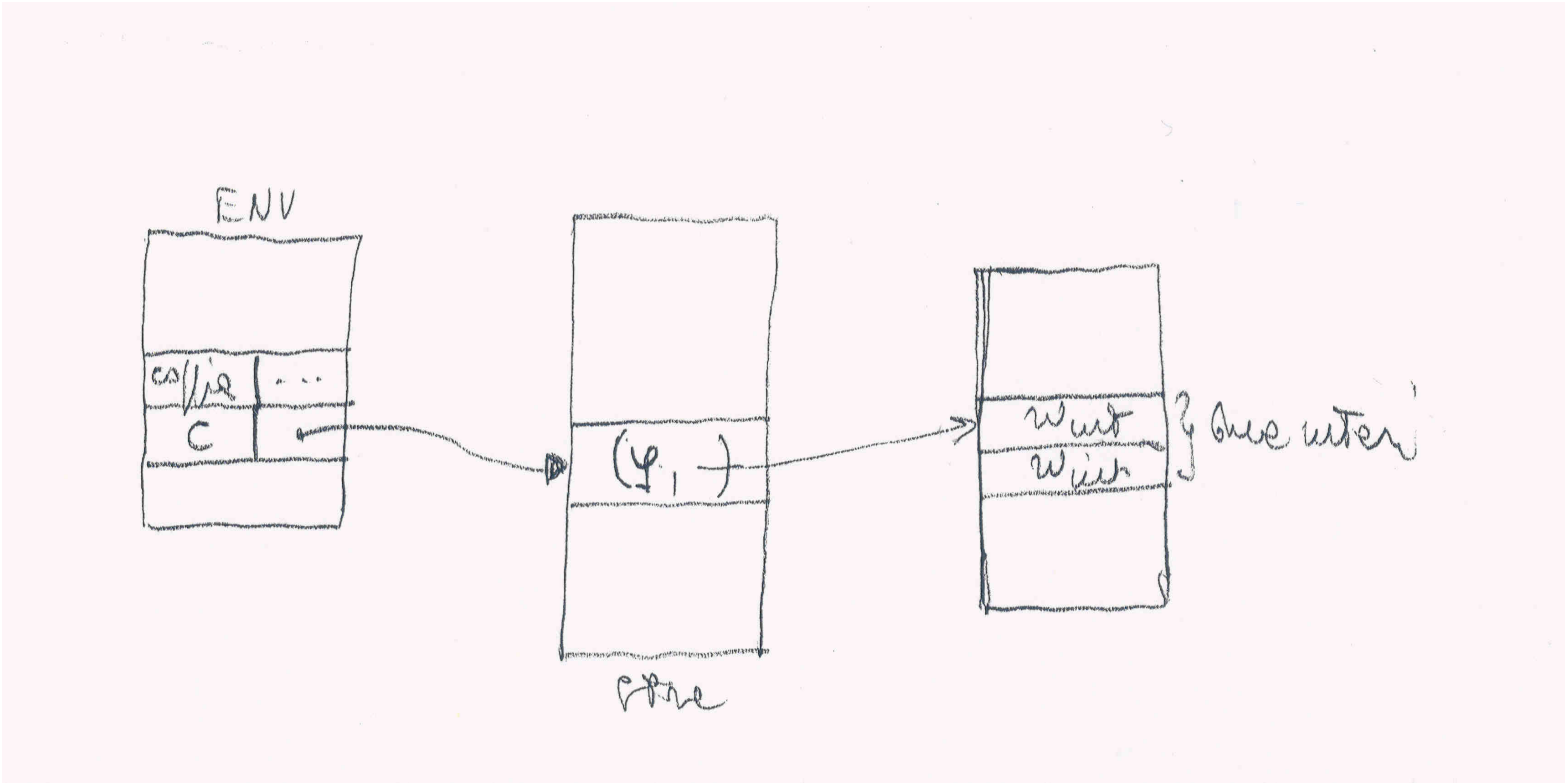


- ed ancora: $\mathtt{c} = \mathtt{Talloc(Coppia)};$

Dichiarazione e Allocazione Statica di Valori Record/3

Implementazione:

- $\mathtt{E_D}(\gamma)$ è implementabile mediante una coppia contenente γ e la prima locazione del blocco allocato per il record
- graficamente: $\mathtt{type\ Coppia} = \mathtt{Rec\ int\ x, int\ y\ End; Coppia} * \mathtt{c;}$.
- ed ancora: $\mathtt{c} = \mathtt{Talloc(Coppia);}$



Tipi e Dati Astratti: API-ADT

- Ortogonalità tra uso e implementazione: l'implementazione non serve per usare il tipo
- Proteggere l'implementazione: inaccessibile a modifiche improprie
- API (specifica) e ADT (definizione) insieme.

Table15.2 – Tipi e ADT(includente API)

Domini Sintattici

$\texttt{D} ::= \ldots \mid \texttt{T I}$ (*naming : Allocazione Statica*)
$\mid \texttt{type I} = \texttt{T}$ (*Definizione di Tipo*)
$\mid \texttt{ADT N} : \overline{\texttt{Z;}}\ \{\overline{\texttt{U;}}\ \overline{\texttt{V;}}\ \overline{\texttt{O;}}\}$
$\mid \ldots$

Domini Sintattici Ausiliari

$\texttt{T} ::= \ldots$ (*tipi anche polimorfi*)
$\texttt{N} ::= \texttt{I} \mid \ldots$ (*nome tipo anche polimorfo*)
$\texttt{Z} ::= \texttt{T I}$ (*tipo e nome operatori visibili*)
$\texttt{U} ::= \texttt{type I} = \texttt{T}$ (*tipi anche ausiliari*)
$\texttt{V} ::= \texttt{I} = \texttt{E}$ (*variabili ausiliarie*)
$\texttt{O} ::= \texttt{I}\ (\overline{\texttt{P}})\ \texttt{C}$ (*operazioni*)

Tipi e Dati Astratti: Sintassi di ADT

Table15.2 – Tipi e ADT
Domini Sintattici
D ::= ... | ADT N : $\overline{\texttt{Z}}$; {$\overline{\texttt{U}}$; $\overline{\texttt{V}}$; $\overline{\texttt{O}}$; | ...

Example

ADT p: `int` × `int` → P mk; P → `int` fst; P → `int` snd;
{`type P = Rec int x, int y End`;
mk(int x, int y){P u = Talloc(P); u.x=x; u.y=y; up(u)};
fst(P x){down(x).x};
snd(P x){down(x).y};
}

- `up`, `down` *trasformano* da V. concreti in astratti e viceversa (risp.)
- La sintassi concreta potrebbe non mostrare l'uso di `up` e `down`
- `up` e `down` potrebbero essere inserite, come avviene per `Val` e `Den`, durante l'analisi statica.

Tipi e Dati Astratti: Semantica di ADT/1

- `up`, `down` *trasformano* da V. concreti in astratti e viceversa (risp.)
- La definizione di `up`, `down` è fondamentale per il significato di un tipo astratto. Consideriamo un ADT per `P`:
  - Le funzioni definite dentro lo ADT quando devono operare su (o modificare) un valore `v` di tipo `P`, di fatto operano sul (modificano il) valore concreto.
  - Se `v` proviene dall'esterno (ad es. è un argomento) allora `v` è trasfomato in `down(v)`.
  - Quando queste funzioni hanno calcolato, il valore `v'` risultante dal calcolo può essere reso all'esterno dell'ADT ma solo come valore astratto `up(v')`
  - Tutte le funzioni all'esterno dell'ADT possono operare solo sui valori astratti, risultanti da `up(v')`.
- `up` e `down` potrebbero essere morfismi tra algebre (quella della segnatura e quella dell'implementazione)

Tipi e Dati Astratti: Semantica di ADT/1fig

- up e down potrebbero essere morfismi tra algebre (quella della segnatura e quella dell'implementazione)

Example

ADT p: $\texttt{int} \times \texttt{int} \rightarrow \texttt{P}\ \texttt{mk};\ \texttt{P} \rightarrow \texttt{int}\ \texttt{fst};\ \texttt{P} \rightarrow \texttt{int}\ \texttt{snd};\texttt{P} \times \texttt{int} \rightarrow ()\ \texttt{m1};$

```
{type P = Rec int x, int y End;
 mk(int x, int y){P u = Talloc(P); u.x=x; u.y=y; up(u)};
 fst(P x){down(x).x};
 snd(P x){down(x).y};
 m1(P x,int v){down(x).x=v;}
}
```

$$\begin{array}{ccccc} \texttt{mk}(3,5) & \rightarrow & \texttt{down} & \rightarrow & \{3,5\} \\ \downarrow & & & & \downarrow \\ \texttt{m1}(\_,0) & & & & \texttt{imp(m1)}(\_,0) \\ \downarrow & & & & \downarrow \\ \texttt{v} & \rightarrow & \texttt{down} & \rightarrow & \{0,5\} \end{array}$$

Tipi e Dati Astratti: Semantica di ADT/2

- `up` e `down` potrebbero essere morfismi tra algebre (quella della segnatura e quella dell'implementazione)
- Daremo una semantica Denotazionale meno astratta e in parte condizionata dalle implementazioni tipicamente utilizzate per gli ADT
- In questa semantica, `up` e `down` sono due funzioni che differiscono tra loro per il modo di accedere al valore di tipo `P`.
- In entrambi i casi l'accesso è basato su una nozione di chiave
- La chiave identifica univocamente un ADT (ed è definita dalla semantica dell'ADT, ovvero l'ambiente γ, definito dall'ADT)
- Un valore è una coppia $(\gamma, \texttt{loc})$, dove γ è la semantica dell'ADT e `loc` la locazione della memoria (heap) dove risiede la struttura allocata per il valore

Tipi e Dati Astratti: Semantica di ADT/2

- In questa semantica, `up` e `down` sono due funzioni che differiscono tra loro per il modo di accedere al valore di tipo `P`.
- Un valore è una coppia $(\gamma, \texttt{loc})$, dove γ è la semantica dell'ADT e `loc` la locazione della memoria (heap) dove risiede la struttura allocata per il valore

  Un valore concreto: $\texttt{c} \equiv (\gamma, \texttt{loc})$,

  Un valore astratto: $\texttt{v} \equiv \texttt{up(c)}$

  $$\texttt{up(c)} = \lambda \texttt{key. if}(\texttt{key} = \gamma, \texttt{c}, \bot_{\texttt{Mem}})$$

  Il valore concreto, dato un astratto (con chiave) γ:

  $$\texttt{down(v)} = \texttt{v}(\gamma)$$

- in nessun caso dal valore astratto si pu accedere `loc`.

Tipi e Dati Astratti: Semantica di ADT/3

- Un valore è una coppia $(\gamma, \texttt{loc})$, dove γ è la semantica dell'ADT e `loc` la locazione della memoria (heap) dove risiede la struttura allocata per il valore
- La funzione `up` rende la coppia accessibile solo se si accede il valore con la chiave attesa.
- La funzione `down` accede un valore con la chiave dell'ADT in cui `down` è invocato.

Nel seguito useremo la seguente notazione per migliorare la leggibilità:
$\rho = \texttt{bind}(\texttt{i}, \texttt{d}, \delta)$ è scritto $(\rho(\texttt{i}) = \texttt{d}) \oplus \delta$ (anche omettendo le parentesi)

Tipi e Dati Astratti: Semantica di ADT/4

Dichiarazione di un binding per una variabile di tipo ADT e definizione di up e down

Table15.3 – ADT

Funzioni Semantiche

$$\mathcal{D}[\![\mathtt{D}]\!]_\rho : \mathtt{Store} \to (\mathtt{Env} \times \mathtt{Store})_\perp$$

$$\mathcal{D}[\![\mathtt{ADT\ N : T_1\ f_1; ...; T_n\ f_n;\ \{\overline{U;}\ \overline{V;}\ \overline{O;}\}\ I}]\!]_\rho(\mathtt{s}) =$$

$$\mathtt{Let}\{\mathtt{Key} = \lambda(\gamma, \mathtt{loc}).\lambda\mathtt{key}.\mathtt{if}(\mathtt{key} = \gamma, (\gamma, \mathtt{loc}), \perp))\}$$

$$\{\sigma_1 = \mathtt{Y}\sigma.\mathcal{D}_E[\![\overline{\mathtt{U;}}]\!]_\sigma \circ \mathcal{D}_E[\![\overline{\mathtt{V;}}]\!]_\sigma \circ \mathcal{D}_E[\![\overline{\mathtt{O;}}]\!]_\sigma \circ$$

$$(\sigma_1(\mathtt{up}) = \mathtt{Key}) \oplus (\sigma_1(\mathtt{down}) = \lambda\mathtt{v}.\mathtt{v}(\sigma_1)) \oplus \rho\}$$

$$\{\rho_1 : \ (\rho_1(\mathtt{f_1}) = \sigma_1(\mathtt{f_1}) \oplus ... \oplus (\rho_n(\mathtt{f_n}) = \sigma_1(\mathtt{f_n}) \oplus \rho\}$$

$$\{(l_1, \mathtt{s_1}) = \mathtt{allocate(s)}\}$$

$$\{\mathtt{s_2} = \mathtt{upd}(\mathtt{l_1}, \mathtt{up}(\sigma_1, \omega), \mathtt{s_1}), \rho_2 = \mathtt{bind}(\mathtt{I}, \mathtt{l_1}, \rho 1)\}$$

$$(\rho 2, \mathtt{s_2})$$

Tipi e Dati Astratti: Semantica di ADT/5

Example

Applichiamo la semantica all'ADT `P` introdotto per un punto cartesiano nell'ipotesi di un codice come sotto:

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