Lezione 29-30
Concetti e Strutture di Livello Avanzato

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

May 14, 2012

Concetti e Strutture di Livello Avanzato

- OO Fundamentals: Costrutti in FJ
- Domini sintattici per Classi, Oggetti
- Domini Semantici: Ambienti come valori
- Funzioni semantiche: Classi, Oggetti, fields, metodi, costruttori
- Implementazione
- Meccanismi addizionali: Polimorfismo generico e di sottotipo
- Meccanismi addizionali: Interfacce e classi anonime
- Meccanismi addizionali: sottoclassi (inner)
- Metodologie: Ereditarietà e super classi
- Metodologie: Tipi Astratti
- Metodologie: Estensione e Riuso di Codice

OO Fundamentals: Costrutti in FJ/1

- Una collezione di classi forma un package, **classes** $\overline{\mathtt{Z}}$ **end**
- Il package fornisce un ottimo supporto per integrare OO con altri paradigmi (quali quelli caratterizzati dai costrutti, HOS incluso, fin quì visti
- Il package può essere nello scope di bindings introdotti con blocchi. La semantica può integrare o separare i paradigmi

Sintassi		
D	::=	... | **classes** $\overline{\mathtt{Z}}$ **end** | ...
C	::=	... | `I := E` | `Call I ()` | ...
E	::=	... | **new** $\mathtt{I}(\overline{\mathtt{E}})$ | **this** | `E.`**super** | $\mathtt{E.G}[(\overline{\mathtt{E}})]$ | ...
Z	::=	**class** `I` [**extends** `N`] $\{\overline{\mathtt{F}}\ \mathtt{K}\ \overline{\mathtt{M}}\}$
N	::=	`I` | **Object**
G	::=	`I`
F	::=	`I = E`
K	::=	**cstr** $= (\overline{\mathtt{P}})\ \mathtt{C}$
M	::=	$\mathtt{I}\ (\overline{\mathtt{P}})\ \mathtt{C}$

OO Fundamentals: Costrutti in FJ/2

- L'opzione [**extends** N] è prevista a livello di sintassi concreta. La mancanza di super classe sottintende l'uso di **extends Object**.

Sintassi		
D	::=	... | **classes** $\overline{\mathtt{Z}}$ **end** | ...
C	::=	... | I := E | Call I () | ...
E	::=	... | **new** I($\overline{\mathtt{E}}$) | **this** | E.**super** | E.G[($\overline{\mathtt{E}}$)] | ...
Z	::=	**class** I [**extends** N] {$\overline{\mathtt{F}}$ K $\overline{\mathtt{M}}$}
N	::=	I | **Object**
G	::=	I
F	::=	I = E
K	::=	**cstr** = ($\overline{\mathtt{P}}$) C
M	::=	I ($\overline{\mathtt{P}}$) C

- la classe contiene una sequenza, anche vuota, di fields, $\overline{\mathtt{F}}$, di metodi, $\overline{\mathtt{M}}$, ed un costruttore, senza nome.

OO Fundamentals: Costrutti in FJ/3

- **super** non è emulabile ed è essenziale per riuso ed ereditarietà.

Sintassi		
D	::=	... | **classes** $\overline{\mathtt{Z}}$ **end** | ...
C	::=	... | `I := E` | `Call I` () | ...
E	::=	... | **new** `I`($\overline{\mathtt{E}}$) | **this** | `E.`**super** | `E.G`[($\overline{\mathtt{E}}$)] | ...
Z	::=	**class** `I` [**extends** `N`] {$\overline{\mathtt{F}}$ `K` $\overline{\mathtt{M}}$}
N	::=	`I` | **Object**
G	::=	`I`
F	::=	`I` = `E`
K	::=	**cstr** = ($\overline{\mathtt{P}}$) `C`
M	::=	`I` ($\overline{\mathtt{P}}$) `C`

- Nell'esempio, il metodo `m` della classe `A` è definito in termini del metodo `m` della superclasse `B`

Example

class `A` extends `B` {... m(..){... this.super.m(...) ...}}

Domini sintattici per Classi, Oggetti

Table 16 – DominiSintattici

Domini Sintattici

$$\mathtt{D} \;::=\; \ldots \mid \textbf{classes}\ \overline{\textbf{class}\ \mathtt{I}\ \textbf{extends}\ \mathtt{N}\ \{\overline{\mathtt{F}}\ \mathtt{K}\ \overline{\mathtt{M}}\}}\ \textbf{end} \mid \ldots$$

$$\mathtt{C} \;::=\; \ldots \mid \mathtt{I} := \mathtt{E} \mid \mathtt{Call\ I}\ () \mid \ldots$$

$$\mathtt{E} \;::=\; \ldots \mid \textbf{new}\ \mathtt{I}(\overline{\mathtt{E}}) \mid \textbf{this} \mid \mathtt{E}.\textbf{super} \mid \mathtt{E.G}(\overline{\mathtt{E}}) \mid \mathtt{E.G} \mid \ldots$$

Domini Sintattici Ausiliari

$$\mathtt{N} \;::=\; \mathtt{I} \mid \textbf{Object} \qquad (\textit{nomi di classe})$$

$$\mathtt{G} \;::=\; \mathtt{I} \qquad (\textit{nomi di fields e metodi})$$

Domini Semantici: Ambienti come valori

Table 16.1 – DominiSemantici	
Domini Semantici	
$\mathtt{Env}, \rho, \delta \equiv \mathtt{I} \rightarrow \mathtt{Den}$	(*Ambienti*)
$\mathtt{Store}, \mathtt{s} \equiv \mathtt{Loc} \rightarrow \mathtt{Mem}$	(*Memoria*)
Domini Semantici Ausiliari	
$\mathtt{Val}, \mathtt{v} ::= \ldots + \mathtt{objectV} + \ldots$	(*Esprimibili*)
$\mathtt{Den}, \mathtt{d} ::= \ldots + \mathtt{objectD} + \ldots$	(*Denotabili*)
$\mathtt{Mem}, \mathtt{d} ::= \ldots + \mathtt{objectM} + \ldots$	(*Memorizzabili*)
Funzioni Ausiliarie	
$\mathtt{O_v} : \mathtt{Env} \rightarrow \mathtt{Val}$	
$\mathtt{O_m} : \mathtt{Env} \rightarrow \mathtt{Mem}$	
$\mathtt{O_d} : \mathtt{Env} \rightarrow \mathtt{Den}$	
$\mathtt{A} : (\mathtt{Val}^n \rightarrow \mathtt{Store} \rightarrow (\mathtt{O}_v \times \mathtt{Store})) \rightarrow \mathtt{Den}$	

- Gli oggetti sono ambienti (non gli stessi dei domini semantici)
- Gli oggetti sono V. esprimibile, memorizzabile, denotabile ($\mathtt{O_v}, \mathtt{O_m}, \mathtt{O_d}$)
- Le classi possono solo costruire oggetti: Quindi, la semantica è un costruttore di Oggetti ($\mathtt{A}$)

Semantica: Package

- Un package racchiude una collezione di classi.
- Ogni classe ha visibilità di sè stessa e di ogni altra classe del package: Questo motiva l'uso del punto fisso

Table16.2 – Package
Funzioni Semantiche
$\mathcal{D}_E[\![\mathtt{D}]\!] : (\mathtt{Env} \times \mathtt{Store}) \rightarrow \mathtt{Env}_\perp$
$\mathcal{D}_E[\![\mathtt{classes}\ \mathtt{C}_1; \ldots; \mathtt{C}_n\ \mathtt{end}]\!]_\rho =$
$\mathtt{Y}\delta\,.\,\mathcal{D}_E[\![\mathtt{C}_1]\!]_\delta \circ \ldots \circ \mathcal{D}_E[\![\mathtt{C}_n]\!]_\delta \circ \rho$

- Un metodo può contenere espressioni per:
  - La creazione;
  - L'accesso e modifica di fields;
  - L'invocazione di metodi

di oggetti di ogni classe del package.

Semantica: Classe/1

- Una classe definisce una funzione per la creazione di oggetti: Funzione d, e corrispondente valore denotabile, A(d).

Table16.3.1 – Classe
Funzioni Semantiche
$\mathcal{D}_E[\![\mathtt{D}]\!] : (\mathtt{Env} \times \mathtt{Store}) \rightarrow \mathtt{Env}$
$\mathcal{D}_E[\![\mathtt{class\ I\ extends\ I}_u\ \{\overline{\mathtt{F}}\ \mathtt{K}\ \overline{\mathtt{M}}\}]\!]_\rho =$
$\mathtt{Let}\{\mathtt{d} = \lambda \overline{\mathtt{v}}_{\mathtt{K}} . \lambda\ \mathtt{s}.\ (\mathtt{O}_{\mathtt{v}}(\rho_{\mathtt{o}}), \mathtt{s}_{\mathtt{o}})\}$
$\mathtt{bind}(\mathtt{I}, \mathtt{A}(\mathtt{d}), \rho)$

- La creazione di un oggetto
  - **può** essere parametrica come in Java e attendersi una lista di parametri $\overline{\mathtt{v}}_{\mathtt{K}}$.
  - **sempre** modifica la memoria, $\mathtt{s}_o$, dove sono conservati gli oggetti
  - **sempre** restituisce un valore esprimibile, $\mathtt{O}_{\mathtt{v}}(\rho_{\mathtt{o}})$, che di fatto è ambiente ρ_o dei fields e dei metodi dell'oggetto creato
- La definizione di $(\mathtt{O}_{\mathtt{v}}(\rho_{\mathtt{o}}), \mathtt{s}_o)$ dipende dal linguaggio.

Semantica: Classe/2

- $(\mathtt{O_v}(\rho_o),\mathtt{s}_o)$ dipende dal linguaggio e può essere complicata.
- Qui richiede tre fasi:
  - creazione di un oggetto della class super, $(\mathtt{o_u},\mathtt{s_u})$
  - estensione dell'oggetto con i field e i method della classe
  - valutazione del codice di inizializzazione (blocco o costruttore come in Java)

Table16.3.2 – Classe
Funzioni Semantiche
$\mathcal{D}[\![\mathtt{D}]\!] : (\mathtt{Env} \times \mathtt{Store}) \rightarrow (\mathtt{Env} \times \mathtt{State})_\perp$
$\quad\mathcal{D}[\![\mathtt{class\ I\ extends\ I_u}\ \{\overline{\mathtt{F}}\ \mathtt{K}\ \overline{\mathtt{M}}\}]\!]_\rho$
$\quad\quad\mathtt{Let}\{\mathtt{d} = \lambda\overline{\mathtt{v}}_\mathtt{K}.\lambda\ \mathtt{s}.$
$\quad\quad\quad\quad\mathtt{Let}\{**\ \textit{creazione oggetto super}\ **\}$
$\quad\quad\quad\quad\quad\{**\ \textit{estensione dell'oggetto}\ **\}$
$\quad\quad\quad\quad\quad\{**\ \textit{valutazione del costruttore}\ **\}$
$\quad\quad\quad\quad(\mathtt{O_v}(\rho_o), \mathtt{s}_o)\}$
$\quad\quad\mathtt{bind}(\mathtt{I}, \mathtt{A}(\mathtt{d}), \rho)$

- creazione di un oggetto della class super, $(\mathtt{o_u}, \mathtt{s_u})$

Table16.3.3 – Classe
Funzioni Semantiche
$\mathcal{D}[\![\mathtt{D}]\!] : (\mathtt{Env} \times \mathtt{Store}) \rightarrow (\mathtt{Env} \times \mathtt{State})_{\perp}$
$\mathcal{D}[\![\mathtt{class\ I\ extends\ I_u}\ \{\overline{\mathtt{F}}\ \mathtt{K}\ \overline{\mathtt{M}}\}]\!]_{\rho}$
$\mathtt{Let}\{\mathtt{d} = \lambda \overline{\mathtt{v}}_{\mathtt{K}} . \lambda\ \mathtt{s}.$
$\mathtt{Let}\{\mathtt{A}(\mathtt{d_u}) = \rho(\mathtt{I_u})\}$
$\{(\mathtt{O_v}(\rho_{\mathtt{u}}), \mathtt{s_u}) = \mathtt{d_u}()(\mathtt{s})\}$
$\{**\ \mathit{estensione\ dell'oggetto}\ **\}$
$\{**\ \mathit{valutazione\ del\ costruttore}\ **\}$
$(\mathtt{O_v}(\rho_{\mathtt{o}}), \mathtt{s_o})\}$
$\mathtt{bind}(\mathtt{I}, \mathtt{A}(\mathtt{d}), \rho)$

- Questo oggetto in Java è un valore accessibile attraverso la "funzione" `super`
- Quì la tralasciamo ma una modifica alle parti che seguono permette di includerla (lasciato come facile esercizio)

Semantica: Classe/4

- estensione dell'oggetto con fields, costruttore e metodi della classe

Table16.3.4 – Classe
Funzioni Semantiche
$\mathcal{D}[\![\mathtt{D}]\!] : (\mathtt{Env} \times \mathtt{Store}) \rightarrow (\mathtt{Env} \times \mathtt{State})_\perp$
$\quad \mathcal{D}[\![\mathtt{class\ I\ extends\ I_u}\ \{\overline{\mathtt{F}}\ \mathtt{K}\ \overline{\mathtt{M}}\}]\!]_\rho$
$\qquad \mathtt{Let}\{\mathtt{d} = \lambda \overline{\mathtt{v}}_\mathtt{K}\,.\,\lambda \mathtt{s}\,.$
$\qquad\qquad \mathtt{Let}\{**\ \textit{creazione oggetto super} : (\mathtt{O_v}(\rho_\mathtt{u}), \mathtt{s_u})\ **\}$
$\qquad\qquad\quad \{\mathtt{Let}\{\lVert \mathtt{A} \rVert \equiv \lambda(\rho', \mathtt{s}')\,.\,\mathcal{D}[\![\mathtt{A}]\!]_{\rho' \circ \rho_\mathtt{u}}(\mathtt{s_u} \circ \mathtt{s}')\}$
$\qquad\qquad\qquad (\rho_o, \mathtt{s}_{\overline{\mathtt{F}}}) = \mathtt{Y}(\rho', \mathtt{s}')\,.\,\lVert \overline{\mathtt{F}} \rVert \circ \lVert \mathtt{K} \rVert \circ \lVert \overline{\mathtt{M}} \rVert\}$
$\qquad\qquad\quad \{**\ \textit{valutazione del costruttore}\ **\}$
$\qquad\qquad (\mathtt{O_v}(\rho_\mathtt{o}), \mathtt{s_o})\}$
$\qquad\quad \mathtt{bind}(\mathtt{I}, \mathtt{A}(\mathtt{d}), \rho)$

- Introduciamo un funzionale $\lVert \mathtt{A} \rVert$ che applicato ad una coppia [di funzioni] $(\rho, \mathtt{s})$ calcola la nuova coppia risultante dalla trasformazione definita da $\mathcal{D}[\![\mathtt{A}]\!]$.

Semantica: Classe/4bis

- estensione dell'oggetto con fields, costruttore e metodi della classe

Table16.3.4 – Classe
Funzioni Semantiche
$\mathcal{D}[\![\mathtt{D}]\!]: (\mathtt{Env} \times \mathtt{Store}) \rightarrow (\mathtt{Env} \times \mathtt{State})_{\perp}$
$\mathcal{D}[\![\mathtt{class\ I\ extends\ I_u}\ \{\overline{\mathtt{F}}\ \mathtt{K}\ \overline{\mathtt{M}}\}]\!]_{\rho}$
$\mathtt{Let}\{\mathtt{d} = \lambda \overline{\mathtt{v}}_{\mathtt{K}}.\lambda \mathtt{s}.$
$\mathtt{Let}\{**\ \textit{creazione oggetto super}: (\mathtt{O_v}(\rho_{\mathtt{u}}), \mathtt{s_u}) **\}$
$\{\mathtt{Let}\{\Vert\mathtt{A}\Vert \equiv \lambda(\rho', \mathtt{s}').\mathcal{D}[\![\mathtt{A}]\!]_{\rho' \circ \rho_{\mathtt{u}}}(\mathtt{s_u} \circ \mathtt{s}')\}$
$(\rho_o, \mathtt{s}_{\overline{\mathtt{F}}}) = \mathtt{Y}(\rho', \mathtt{s}').\Vert\overline{\mathtt{F}}\Vert \circ \Vert\mathtt{K}\Vert \circ \Vert\overline{\mathtt{M}}\Vert\}$
$\{**\ \textit{valutazione del costruttore} **\}$
$(\mathtt{O_v}(\rho_{\mathtt{o}}), \mathtt{s_o})\}$
$\mathtt{bind}(\mathtt{I}, \mathtt{A}(\mathtt{d}), \rho)$

- Vogliamo il fixpoint della composizione di tali trasformazioni in modo tale che tutti i bindings (e locazioni allocate) generati siano visibili in tutte le definizioni di fields, metodi e costruttore .

Semantica: Classe/4ter

- estensione dell'oggetto con fields, costruttore e metodi della classe

Table16.3.4 – Classe
Funzioni Semantiche
$\mathcal{D}[\![\mathtt{D}]\!] : (\mathtt{Env} \times \mathtt{Store}) \to (\mathtt{Env} \times \mathtt{State})_\perp$
$\quad \mathcal{D}[\![\mathtt{class\ I\ extends\ I_u}\ \{\overline{\mathtt{F}}\ \mathtt{K}\ \overline{\mathtt{M}}\}]\!]_\rho$
$\qquad \mathtt{Let}\{\mathtt{d} = \lambda \overline{\mathtt{v}}_\mathtt{K}.\lambda \mathtt{s}.$
$\qquad\qquad \mathtt{Let}\{**\ \textit{creazione oggetto super} : (\mathtt{O_v}(\rho_\mathtt{u}), \mathtt{s_u})**\}$
$\qquad\qquad\quad \{\mathtt{Let}\{\lVert\mathtt{A}\rVert \equiv \lambda(\rho', \mathtt{s}').\mathcal{D}[\![\mathtt{A}]\!]_{\rho' \circ \rho_\mathtt{u}}(\mathtt{s_u} \circ \mathtt{s}')\}$
$\qquad\qquad\qquad (\rho_o, \mathtt{s}_{\overline{\mathtt{F}}}) = \mathtt{Y}(\rho', \mathtt{s}').\lVert\overline{\mathtt{F}}\rVert \circ \lVert\mathtt{K}\rVert \circ \lVert\overline{\mathtt{M}}\rVert\}$
$\qquad\qquad\quad \{**\ \textit{valutazione del costruttore}**\}$
$\qquad\qquad (\mathtt{O_v}(\rho_\mathtt{o}), \mathtt{s_o})\}$
$\qquad\quad \mathtt{bind}(\mathtt{I}, \mathtt{A}(\mathtt{d}), \rho)$

- La semantica data da $\mathcal{D}[\![]\!]$ per i fields potrebbe utilizzare quella data per la dichiarazione di variabile procedurale.
- Ovviamente, con le espressioni di inizializzazione estese con le nuove espressioni su oggetti.

Semantica: Classe/4quater

- estensione dell'oggetto con fields, costruttore e metodi della classe

Table16.3.4 – Classe
Funzioni Semantiche
$\mathcal{D}[\![\mathtt{D}]\!] : (\mathtt{Env} \times \mathtt{Store}) \to (\mathtt{Env} \times \mathtt{State})_\perp$
$\quad \mathcal{D}[\![\mathtt{class\ I\ extends\ I_u}\ \{\overline{\mathtt{F}}\ \mathtt{K}\ \overline{\mathtt{M}}\}]\!]_\rho$
$\qquad \mathtt{Let}\{\mathtt{d} = \lambda \overline{\mathtt{v}}_\mathtt{K}.\lambda \mathtt{s}.$
$\qquad\qquad \mathtt{Let}\{**\ \textit{creazione oggetto super} : (\mathtt{O_v}(\rho_\mathtt{u}), \mathtt{s_u})\ **\}$
$\qquad\qquad\quad \{\mathtt{Let}\{\Vert\mathtt{A}\Vert \equiv \lambda(\rho', \mathtt{s}').\mathcal{D}[\![\mathtt{A}]\!]_{\rho' \circ \rho_\mathtt{u}}(\mathtt{s_u} \circ \mathtt{s}')\}$
$\qquad\qquad\qquad (\rho_o, \mathtt{s}_{\overline{\mathtt{F}}}) = \mathtt{Y}(\rho', \mathtt{s}').\Vert\overline{\mathtt{F}}\Vert \circ \Vert\mathtt{K}\Vert \circ \Vert\overline{\mathtt{M}}\Vert\}$
$\qquad\qquad\quad \{**\ \textit{valutazione del costruttore}\ **\}$
$\qquad\qquad (\mathtt{O_v}(\rho_\mathtt{o}), \mathtt{s_o})\}$
$\qquad \mathtt{bind}(\mathtt{I}, \mathtt{A}(\mathtt{d}), \rho)$

- In aggiunta a fields e metodi, ρ_o contiene anche un costruttore
- La semantica dei metodi potrebbe utilizzare quella delle procedure,
- utilizzando però una diversa invocazione come vedremo più avanti.

Semantica: Classe/5

- valutazione del codice di inizializzazione (blocco o, costruttore come in Java)

Table16.3.5 – Classe
Funzioni Semantiche
$\mathcal{D}[\![\mathtt{D}]\!] : (\mathtt{Env} \times \mathtt{Store}) \to (\mathtt{Env} \times \mathtt{State})_\perp$
$\mathcal{D}[\![\mathtt{class\ I\ extends\ I_u}\ \{\overline{\mathtt{F}}\ \mathtt{K}\ \overline{\mathtt{M}}\}]\!]_\rho$
$\mathtt{Let}\{\mathtt{d} = \lambda \overline{\mathtt{v}}_\mathtt{K}.\lambda \mathtt{s}.$
$\mathtt{Let}\{**\ \textit{creazione oggetto super} : (\mathtt{O_v}(\rho_\mathtt{u}), \mathtt{s_u}) **\}$
$\{**\ \textit{estensione dell'oggetto} : (\rho_o, \mathtt{s}_{\overline{\mathtt{F}}}) **\}$
$\{\mathtt{s}_o = \rho_o(\mathtt{cstr})(\overline{\mathtt{v}}_\mathtt{K})(\mathtt{s}_{\overline{\mathtt{F}}})\}$
$(\mathtt{O_v}(\rho_\mathtt{o}), \mathtt{s_o})\}$
$\mathtt{bind}(\mathtt{I}, \mathtt{A}(\mathtt{d}), \rho)$

- Quì, i costruttori sono senza parametri: $(\overline{\mathtt{v}}_\mathtt{K}) = ()$.
- In presenza di parametri, $\overline{\mathtt{v}}_\mathtt{K}$ deve essere partizionato in due sottosequenze: la prima deve essere applicata al costruttore di $\mathtt{I_u}$, in accordo al numero di argomenti attesi.

Semantica: Classe/6

- La definizione completa

Table16.3 – Classe
Funzioni Semantiche
$\mathcal{D}[\![\mathtt{D}]\!] : (\mathtt{Env} \times \mathtt{Store}) \to (\mathtt{Env} \times \mathtt{State})_{\perp}$
$\mathcal{D}[\![\mathtt{class\ I\ extends\ I_u}\ \{\overline{\mathtt{F}}\ \mathtt{K}\ \overline{\mathtt{M}}\}]\!]_{\rho}$
$\mathtt{Let}\{\mathtt{d} = \lambda \overline{\mathtt{v}}_{\mathtt{K}}.\lambda \mathtt{s}.$
$\mathtt{Let}\{\mathtt{A}(\mathtt{d_u}) = \rho(\mathtt{I_u})\}$
$\{(\mathtt{O_v}(\rho_{\mathtt{u}}), \mathtt{s_u}) = \mathtt{d_u}()(\mathtt{s})\}$
$\{\mathtt{Let}\{\Vert\mathtt{A}\Vert \equiv \lambda(\rho', \mathtt{s}').\mathcal{D}[\![\mathtt{A}]\!]_{\rho' \circ \rho_{\mathtt{u}}}(\mathtt{s_u} \circ \mathtt{s}')\}$
$(\rho_o, \mathtt{s}_{\overline{\mathtt{F}}}) = \mathtt{Y}(\rho', \mathtt{s}').\Vert\overline{\mathtt{F}}\Vert \circ \Vert\mathtt{K}\Vert \circ \Vert\overline{\mathtt{M}}\Vert\}$
$\{\mathtt{s}_o = \rho_o(\mathtt{cstr})(\overline{\mathtt{v}}_{\mathtt{K}})(\mathtt{s}_{\overline{\mathtt{F}}})\}$
$(\mathtt{O_v}(\rho_{\mathtt{o}}), \mathtt{s_o})\}$
$\mathtt{bind}(\mathtt{I}, \mathtt{A}(\mathtt{d}), \rho)$

- Manca il self-reference `this`: A cosa Serve? Può essere evitato (come e dove)? Come lo si può aggiungere?

- Manca il self-reference **this**: A cosa Serve? Può essere evitato (come e dove)? Come lo si può aggiungere?

Example

Table16.3 – Classe con SelfReference

Funzioni Semantiche

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

$$\mathcal{D}[\![\mathtt{class\ I\ extends\ I_u}\ \{\overline{\mathtt{F}}\ \mathtt{K}\ \overline{\mathtt{M}}\}]\!]_\rho$$

$$\begin{array}{l}
\mathtt{Let}\{\mathtt{d} = \lambda\overline{\mathtt{v}}_\mathtt{K}.\lambda\mathtt{s}. \\
\quad \mathtt{Let}\{\mathtt{A}(\mathtt{d_u}) = \rho(\mathtt{I_u})\} \\
\quad\quad \{(\mathtt{O_v}(\rho_\mathtt{u}), \mathtt{s_u}) = \mathtt{d_u}()(\mathtt{s})\} \\
\quad\quad \{\mathtt{Let}\{\|\mathtt{A}\| \equiv \lambda(\rho', \mathtt{s}').\mathcal{D}[\![\mathtt{A}]\!]_{\rho' \circ \rho_\mathtt{u}}(\mathtt{s_u} \circ \mathtt{s}'); \\
\quad\quad\quad \mathtt{self} \equiv \lambda(\rho', \mathtt{s}').(\mathtt{bind}(\mathtt{this}, \mathtt{O_v}(\rho'), \rho'), \mathtt{s}')\} \\
\quad\quad\quad (\rho_o, \mathtt{s}_{\overline{\mathtt{F}}}) = \mathtt{Y}(\rho, \mathtt{s}).\|\overline{\mathtt{F}}\| \circ \|\mathtt{K}\| \circ \|\overline{\mathtt{M}}\| \circ \mathtt{self}\} \\
\quad\quad \{\mathtt{s}_o = \rho_o(\mathtt{cstr})(\overline{\mathtt{v}}_\mathtt{K})(\mathtt{s}_{\overline{\mathtt{F}}})\} \\
\quad\quad (\mathtt{O_v}(\rho_o), \mathtt{s}_o)\} \\
\mathtt{bind}(\mathtt{I}, \mathtt{A}(\mathtt{d}), \rho)
\end{array}$$

Semantica: Classe/6ter

Manca **super**: A cosa Serve? Può essere evitato (come e dove)? Come lo si può aggiungere?

Example

Table16.3 – Classe con SelfReference e Super
Funzioni Semantiche
$\mathcal{D}[\![\mathtt{D}]\!] : (\mathtt{Env} \times \mathtt{Store}) \rightarrow (\mathtt{Env} \times \mathtt{State})_\perp$
$\quad \mathcal{D}[\![\mathtt{class\ I\ extends\ I_u\ \{\overline{F}\ K\ \overline{M}\}}]\!]_\rho$
$\quad\quad \mathtt{Let}\{\mathtt{d} = \lambda \overline{\mathtt{v}}_{\mathtt{K}}.\lambda \mathtt{s}.$
$\quad\quad\quad\quad \mathtt{Let}\{\mathtt{A}(\mathtt{d_u}) = \rho(\mathtt{I_u})\}$
$\quad\quad\quad\quad\quad \{(\mathtt{O_v}(\rho_{\mathtt{u}'}), \mathtt{s_u}) = \mathtt{d_u}()(\mathtt{s})\}$
$\quad\quad\quad\quad\quad \{\rho_{\mathtt{u}} = \mathtt{bind}(\mathtt{super}, \mathtt{O_v}(\rho_{\mathtt{u}'}), \rho_{\mathtt{u}'})\}$
$\quad\quad\quad\quad\quad \{\mathtt{Let}\{\lVert\mathtt{A}\rVert \equiv \lambda(\rho', \mathtt{s}').\mathcal{D}[\![\mathtt{A}]\!]_{\rho' \circ \rho_{\mathtt{u}}}(\mathtt{s_u} \circ \mathtt{s}');$
$\quad\quad\quad\quad\quad\quad\quad \mathtt{self} \equiv \lambda(\rho', \mathtt{s}').(\mathtt{bind}(\mathtt{this}, \mathtt{O_v}(\rho'), \rho'), \mathtt{s}')\}$
$\quad\quad\quad\quad\quad\quad (\rho_o, \mathtt{s}_{\overline{\mathtt{F}}}) = \mathtt{Y}(\rho, \mathtt{s}).\lVert\overline{\mathtt{F}}\rVert \circ \lVert\mathtt{K}\rVert \circ \lVert\overline{\mathtt{M}}\rVert \circ \mathtt{self}\}$
$\quad\quad\quad\quad\quad \{\mathtt{s}_o = \rho_o(\mathtt{cstr})(\overline{\mathtt{v}}_{\mathtt{K}})(\mathtt{s}_{\overline{F}})\}$
$\quad\quad\quad\quad (\mathtt{O_v}(\rho_{\mathtt{o}}), \mathtt{s_o})\}$
$\quad\quad \mathtt{bind}(\mathtt{I}, \mathtt{A}(\mathtt{d}), \rho)$

Semantica: Classe Object

- La classe Object richiede una definizione speciale
- La classe Object è l'unica classe che non ha superclasse
- Quì assumiamo che non abbia bindings (in Java ha quelli dei fields e dei metodi primitivi)

Table16.4 – Classe Object
Funzioni Semantiche
$\mathcal{D}[\![\mathtt{D}]\!] : (\mathtt{Env} \times \mathtt{Store}) \to (\mathtt{Env} \times \mathtt{State})_\perp$
$\mathcal{D}[\![\mathtt{class\ Object}\ \{\}]\!]_\rho$
$\mathtt{Let}\{\mathtt{d} = \lambda().\lambda\ \mathtt{s}.(\mathtt{O_v}(\lambda\mathtt{x}.\mathtt{x}), \mathtt{s})\}$
$\mathtt{bind}(\mathtt{I}, \mathtt{A}(\mathtt{d}), \rho)$

- La semantica usa l'ambiente vuoto $\lambda\mathtt{x}.\mathtt{x}$ (in accordo alle osservazioni fatte sul dominio degli ambienti)

Implementazione: Classe/1

- Come è implementata?

Table16.3 – Classe
Funzioni Semantiche
$\mathcal{D}[\![\mathtt{D}]\!] : (\mathtt{Env} \times \mathtt{Store}) \to (\mathtt{Env} \times \mathtt{State})_\perp$
$\mathcal{D}[\![\mathtt{class\ I\ extends\ I_u}\ \{\overline{\mathtt{F}}\ \mathtt{K}\ \overline{\mathtt{M}}\}]\!]_\rho$
$\mathtt{Let}\{\mathtt{d} = \lambda\overline{\mathtt{v}}_\mathtt{K}.\lambda\mathtt{s}.$
$\mathtt{Let}\{\mathtt{A}(\mathtt{d_u}) = \rho(\mathtt{I_u})\}$
$\{(\mathtt{O_v}(\rho_\mathtt{u}), \mathtt{s_u}) = \mathtt{d_u}()(\mathtt{s})\}$
$\{\mathtt{Let}\{\Vert\mathtt{A}\Vert \equiv \lambda(\rho', \mathtt{s}').\mathcal{D}[\![\mathtt{A}]\!]_{\rho' \circ \rho_\mathtt{u}}(\mathtt{s_u} \circ \mathtt{s}')\}$
$(\rho_o, \mathtt{s}_{\overline{\mathtt{F}}}) = \mathtt{Y}(\rho', \mathtt{s}').\Vert\overline{\mathtt{F}}\Vert \circ \Vert\mathtt{K}\Vert \circ \Vert\overline{\mathtt{M}}\Vert\}$
$\{\mathtt{s}_o = \rho_o(\mathtt{cstr})(\overline{\mathtt{v}}_\mathtt{K})(\mathtt{s}_{\overline{\mathtt{F}}})\}$
$(\mathtt{O_v}(\rho_\mathtt{o}), \mathtt{s_o})\}$
$\mathtt{bind}(\mathtt{I}, \mathtt{A}(\mathtt{d}), \rho)$

- Dipende dal supporto su cui implementiamo: JVM (Java Virtual Machine) oppure P-Machine (Pascal) oppure 3-address code

Implementazione: Classe/2

- Dipende dal supporto su cui implementiamo: P-Machine (Pascal)
  - Utilizziamo activation records: AR e memoria statica, memoria dinamica a stack (per le invocazioni, come vediamo più avanti) e a heap per gli oggetti (come vediamo dopo)
  - Il frame di un AR contiene l'ambiente con le nuove denotazioni per le classi di un package.

Example

Si completi opportunamente il seguente package

```
classes class A extends Object{
        int x = this.y.x + 2;
        C y = null;
        cstr(C u) {y=u;};
        int add(int u){this.x=2*this.x+u;};
        }
```

e se ne mostri, graficamente, la struttura degli AR (limitatamente al solo componente frame).

- Il frame di un AR contiene l'ambiente con le nuove denotazioni per le classi di un package.

Poniamo completare con la classe C di cui omettiamo la cui definizione è ininfluente ai fini dell'esercizio

class C extends Object{...}
end

Il frame e l'AR hanno la seguente forma.

...		R.I.	CS(PI)	C.d.
A	A(d_A)			RET
C	A(d_C)			VAL
				PC

dove d_A e d_C sono le funzioni calcolate dalla semantica delle due classi

Semantica: Creazione di oggetti, invocazione, selezione

- La creazione è un'invocazione della funzione associata alla classe
- Utilizziamo la semantica dell'invocazione *completa* di procedura (vedi esercizio sulla definizione completa)

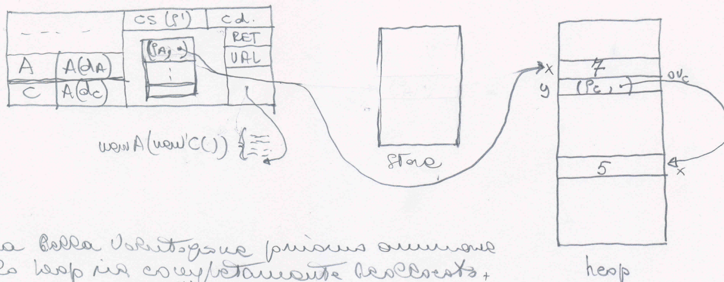
Table16.5.1 – Creazione di oggetti
Funzioni Semantiche
$\mathcal{E}[\![\mathtt{E}]\!]_\rho : \mathtt{Store} \to (\mathtt{Val} \times \mathtt{Store})_\perp$
$\mathcal{E}[\![\mathtt{new\ I}(\mathtt{A}_1 \ldots \mathtt{A}_n)]\!]_\rho(\mathtt{s}) =$
$\mathtt{Let}\{\mathtt{A}(\mathtt{d}) = \rho(\mathtt{I}), ((v_1 \ldots v_n), \mathtt{s}_n) = \mathcal{T}[\![(\mathtt{A}_1 \ldots \mathtt{A}_n)]\!]_\rho(\mathtt{s})\}$
$\mathtt{d}(v_1 \ldots v_n)(\mathtt{s}_n)$

Example

Si mostri, graficamente, la struttura della memoria prima e dopo la valutazione di: `new A(new C())`
nell'AR ottenuto dall'esempio del precedente lucido sui package.

Implementazione: Creazione/sol

- La creazione è un'invocazione della funzione associata alla classe



Semantica: Creazione di oggetti, invocazione, selezione

- È un'invocazione della funzione associata alla classe
- Utilizziamo la semantica dell'invocazione *completa* di procedura (vedi esercizio sulla definizione completa)

Table16.5.2 – Invocazione di metodo

Funzioni Semantiche

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

$$\mathcal{E}[\![\mathtt{E.I}(\mathtt{A}_1 \ldots \mathtt{A}_n)]\!]_\rho(\mathtt{s}) =$$

$$\mathtt{Let}\{(\mathtt{O_v}(\rho_0), \mathtt{s}_0) = \mathcal{E}[\![\mathtt{E}]\!]_\rho(\mathtt{s})\}$$

$$\{((v_1 \ldots v_n), \mathtt{s}_n) = \mathcal{T}[\![(\mathtt{A}_1 \ldots \mathtt{A}_n)]\!]_\rho(\mathtt{s}_0),\ \mathtt{F}(\mathtt{f}) = \rho_0(\mathtt{I})\}$$

$$\mathtt{f}(v_1 \ldots v_n)(\mathtt{s}_n)$$

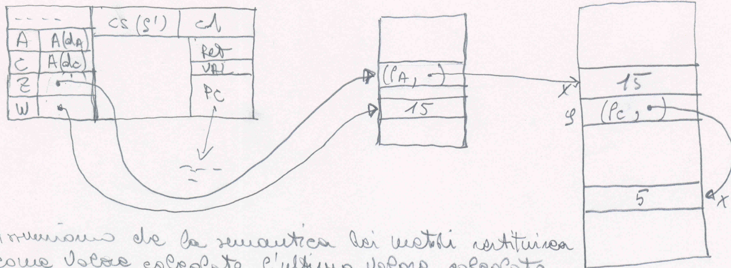
Example

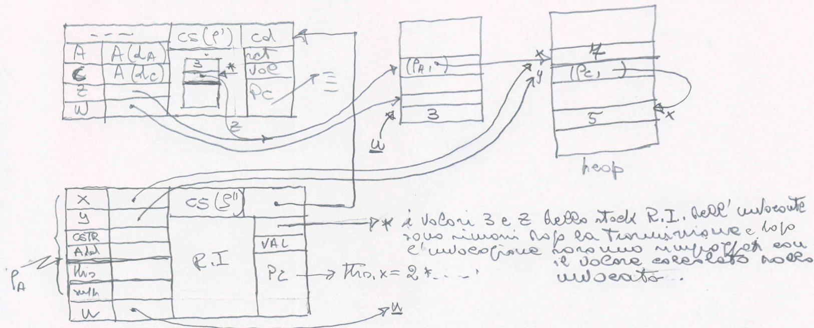
Si mostri, graficamente, la struttura della memoria prima e dopo la valutazione di:

```
A z = new A(new C());
int w = z.add(3);
```

nell'AR ottenuto dall'esempio del precedente lucido sui package

- La creazione è un'invocazione della funzione associata alla classe




heap
R.I
VAL
CSTR
3
5

Semantica: Creazione di oggetti, invocazione, selezione

- È un'accesso ai valori denotati da identificatori nell'ambiente definito dall'oggetto.
- Utilizziamo la semantica del binding

Table16.5.3 – selezione super e fields
Funzioni Semantiche
$\mathcal{E}[\![\mathtt{E}]\!]_\rho$: $\mathtt{Store} \rightarrow (\mathtt{Val} \times \mathtt{Store})_\perp$
$\mathcal{E}[\![\mathtt{E.super}]\!]_\rho(\mathtt{s}) = \mathtt{Let}\{(\mathtt{O_v}(\rho_0), \mathtt{s}_0) = \mathcal{E}[\![\mathtt{E}]\!]_\rho(\mathtt{s})\}$ $(\rho_0(\mathtt{super}), \mathtt{s}_0)$
$\mathcal{E}[\![\mathtt{E.I}]\!]_\rho(\mathtt{s}) = \mathtt{Let}\{(\mathtt{O_v}(\rho_0), \mathtt{s}_0) = \mathcal{E}[\![\mathtt{E}]\!]_\rho(\mathtt{s})\}$ $(\mathtt{MV}(\mathtt{look}(\rho_0(\mathtt{I}), \mathtt{s}_0)), \mathtt{s}_0)$

Example

Si mostri, graficamente, la struttura della memoria prima e dopo la valutazione di:...

Semantica: Creazione di oggetti, invocazione, selezione

Example

Si mostri, graficamente, la struttura della memoria prima e dopo la valutazione di:

```
A z = new B(new C());
int w = z.super.x;
```

nell'AR ottenuto dal seguente package:

```
classes class A extends Object{
            int x = this.y.x + 2;
            C y = null;
            cstr(C u) {y=u;};
            int add(int u){this.x=2*this.x+u;};}
        class B extends A{
            int x = 0;
            cstr() {x=10;};
            int add(int u){this.x=u;};}
        class C extends Object{int y = 7; cstr(){};}
end
```

Meccanismi Addizionali: Polimorfismo generico e di sottotipo

- Polimorfismo di sottotipo. Relazione esplicita `extends`
- Nell'esempio, A è sottotipo di B: Gli oggetti possono essere utilizzati come oggetti B

Example

Si completi opportunamente il seguente package Java

```
class A<T extends B> extends D{
        T x = this.y.x.m1(2);
        C<T> y = new C<T>;
        A(C<T> u) {y=u;};
        int add(int u){this.x.m2(2*this.x.m3(u));};
        }
```

Si mostri una definizione per i metodi m1, m2, m3 utilizzati nell'esempio.

- Polimorfismo generico: T è quantificata universalmente (sui tipi che precedono B)

Meccanismi Addizionali: Interfacce e classi anonime

- Interfacce: Astrazioni di dati (API e ADT) e non solo
- Combinate con classi anonime per generare dinamicamente oggetti di nuovi tipi

Example

Un'interfaccia, oggetti di un nuovo tipo, uso di oggetto

```
interface Fun<T1, T2> {T1 apply(T2 x)}
        ...
Fun<int, int>sqr= new Fun<int, int>(){
        apply(int x){return x*x;}};
Fun<int, int>fact= new Fun<int, int>(){
        apply(int x){return (x==0)?1:x*apply(x-1);}};
        ...
... sqr.apply(5)...
```

Meccanismi Addizionali: sottoclassi

- Diamo per scontato esistenza di sottoclassi ma potrebbero non esserci
- Perderemmo molto sia in termini di meccanismi sia in termini di metodologie di programmazione
  - polimorfismo di sottotipo
  - ereditarietà
  - riuso di codice

Metodologie: Ereditarietà e Super classi

Example

```
public class TwoPoint{
  double x; double y;
  public Point (double a, double b){x=a; y=b};
  public double projectionX(){return x}
...
  public double distance (double a, double b){
    double xx = x-a; double yy= y-b;
    return math.sqrt(xx*xx + yy*yy);
}
public class ThreePoint extends TwoPoint{
  double z;
  // ** ma prima e seconda componente sono ereditate
  public Point (double a, double b, double c){
    // **da ridefinire **};
  // ** ma projectionX è ereditata ed opera bene
  public double distance(double a, double b, double c){
    // **da ridefinire **};
}
```

Metodologie: Tipi Astratti

- un API per il tipo relazione

Example

```
public abstract class Relazione <A,B>{
        public abstract Relazione ();
        public abstract void add (A x, B y);
        public abstract void remove (A x, B y);
        public abstract LinkedList <A> getUno(B y);
}
```

- il tipo definito è molto diverso dall'API funzionale vista in Caml: perchè?

Metodologie: Estensione e Riuso di Codice

Example

```
public class TwoPoint{
  double x; double y;
  public Point (double a, double b){x=a; y=b};
  public double ProjectionX(){return x}
...
  public double distance (double a, double b){
    double xx = x-a; double yy= y-b;
    return math.sqrt(xx*xx + yy*yy);
}
public class ThreePoint extends TwoPoint{
  double z;
  public Point (double a, double b, double c){
    super(a,b); z=c;};
  public double distance(double a, double b, double c){
        ...
}
```