Lezione 7-8

February 29, 2012

Lezione 7-8

February 29, 2012

Conoscenza dei costrutti di un Linguaggio

La conoscenza dei costrutti di un Linguaggio di Programmazione avviene attraverso l'uso di vari strumenti la cui disponibilità però, dipende dallo specifico linguaggio.

- **Definizione Formale** Sintassi e Semantica e Macchina Astratta;
- **Esecutori** Compilatori, Interpreti con cui fare esperienza e testing di programmi del linguaggio;
- **Testi di Programmazione del Linguaggio**: Costruzione di programmi per algoritmi di uso generale e/o specifico per le applicazioni del linguaggio;
- **Manuali di Riferimento e Uso** Descrizione dei singoli costrutti, del comportamento dell'esecutore, delle combinazioni tipiche dei costrutti per formare programmi.

Paradigmi di Programmazione/1

Lo studio dei paradigmi conduce e richiede la conoscenza dei costrutti dei Linguaggi di Programmazione, ma:

- **Necessità di Confronto** Lo studio dei paradigmi ci conduce a costrutti diversi per semantica e realizzazione ma con comportamenti talvolta simili nello specifico paradigma (ad esempio: costante procedurale e parametro funzionali, iteratore procedurale e iteratore funzionale, etc.).
- **Omogeneità di Definzione** A parte la sintassi (grammatiche EBNF), gli strumenti, quando presenti, utilizzati nelle definizioni fanno ricorso a formalismi diversi (ad esempio: semantica operazionale, invece di denotazionale, etc.)
- **Macchine Astratte** Non sempre disponibili o troppo impegnative (Macchine Virtuali: P-code Machine, WAM, JVM)

Paradigmi di Programmazione/2

Per questa ragione nella conoscenza dei costrutti utilizzeremo:

- **Semantica Denotazionale** che risulta sufficientemente astratta da non influenzare l'effettiva implementazione del costrutto;
- **Modelli o Schemi di Implementazione** invece di effettive realizzazioni della Macchina Astratta;
- **Metodologie di Programmazione** in cui collocheremo l'uso e il ruolo dei vari costrutti.
- **Compilatori e/o Interpreti** con cui sperimenteremo e faremo testing di tutte le costruzioni interessanti.

Semantica Denotazionale

Le strutture base sono:

- **Domini Sintattici**
- **Domini Semantici**
- **Funzioni Semantiche**

Queste 3 entità caratterizzano il linguaggio e ne permettono un **confronto analitico**

Domini Sintattici

Table 1

Domini Sintattici

D	::=	`Proc I() C;` | `D D` | ...	(*Dichiarazioni*)
C	::=	`{D C}` | `I := E` | `Call I ()` | ...	(*Comandi*)
E	::=	`I` | `VL` | ...	(*Espressioni*)
VL	::=	...	(*Literals*)

Example

Si elenchino i costruttori, indicandone la segnatura, utilizzati:
$\{\_\ \_\} : \mathtt{D} \times \mathtt{C} \rightarrow \mathtt{C}$)

Domini Semantici/1

Table 2

Domini Semantici

$$
\begin{array}{lcl}
\mathtt{Env}, \rho, \delta & \equiv & \mathtt{I} \longrightarrow \mathtt{Den} \qquad\qquad (\textit{Ambienti}) \\
 & & \textit{operazioni di } \mathtt{Env}: \\
 & & \mathtt{bind}: \ \mathtt{I} \times \mathtt{Den} \times \mathtt{Env} \longrightarrow \mathtt{Env} \\
 & & \quad \mathtt{bind}(\mathtt{I}, \mathtt{d}, \rho) \equiv \lambda \mathtt{x}.\ \mathtt{if}((\mathtt{x}\ \textit{eq}\ \mathtt{I}), \mathtt{d}, \rho(\mathtt{x})) \\
 & & \mathtt{find}: \ \mathtt{I} \times \mathtt{Env} \longrightarrow \mathtt{Den} \\
 & & \quad \mathtt{find}(\mathtt{I}, \rho) \equiv \rho(\mathtt{I}) \\
 & & \mathtt{empty}: \mathtt{Env} \\
 & & \quad \mathtt{empty} \equiv \lambda \mathtt{x}.\ \mathtt{x}
\end{array}
$$

Domini Semantici/2

Table 2.continua

$\texttt{Store}, \texttt{s}$	$\equiv$	...	(*Memoria*)
		operazioni di $\texttt{Store}$:	
		$\texttt{upd} : \texttt{Loc} \times \texttt{Mem} \times \texttt{Store} \rightarrow \texttt{Store}$	
		$\texttt{look} : \texttt{Loc} \times \texttt{Store} \rightarrow \texttt{Mem}$	
$\texttt{State}$	$::=$	...	(*Stato*)

Domini Semantici Ausiliari

$\texttt{Val}, \texttt{v}$	$::=$	...	(*Esprimibili*)
$\texttt{Den}, \texttt{d}$	$::=$	...	(*Denotabili*)
$\texttt{Mem}, \texttt{d}$	$::=$	...	(*Memorizzabili*)
$\texttt{Loc}, \texttt{l}$	$::=$	...	(*Locazioni*)
$\texttt{Input}$	$::=$	...	(I/O : *Input*)
$\texttt{Output}$	$::=$	...	(I/O : *Output*)

Domini Semantici/3

Example

Applichiamo al seguente caso: Dato un programma di un linguaggio noto, si devono individuare i valori denotabili, memorizzabili, ed esprimibili utilizzati nel programma. Si deve motivare ogni scelta.

Example

Applichiamo al seguente caso: In C gli array sono valori denotabili e memorizzabili ma non esprimibili. In Java invece, sono anche esprimibili. Mostriamo un codice che non si può esprimere, per tale ragione, in C e che, invece, può essere espresso in Java.

Domini Semantici/3

Example

Confrontiamo C, Ocaml, Java mediante una tabella la cui prima colonna, intitolata valori, contiene tutti i valori dei 3 linguaggi senza distinzione. Le colonne successive sono intitolate denotabili, memorizzabili ed esprimibili rispettivamente, e contengono, in corrispondenza alle varie righe, la lista dei linguaggi aventi tale valore in tale classe.

Funzioni Semantiche/1

Table 3

Funzioni Semantiche

$\mathcal{D} : \mathtt{D} \rightarrow \mathtt{Env} \rightarrow \mathtt{Env}$ (*Significato delle Dichiarazioni*)

$\mathcal{M} : \mathtt{C} \rightarrow \mathtt{Env} \rightarrow \mathtt{State} \rightarrow \mathtt{State}$ (*Significato dei Comandi*)

$\mathcal{E} : \mathtt{E} \rightarrow \mathtt{Env} \rightarrow \mathtt{State} \rightarrow \mathtt{Val}$ (*Significato delle Espressioni*)

Domini Ausiliari

$\mathtt{VL} ::= \mathtt{Int} + \mathtt{Char}$ (*Valori Literals : unione disgiunta*)

$\mathtt{Int} ::= \ldots$ (*Valori Literals per interi*)

$\mathtt{Char} ::= \ldots$ (*Valori Literals per caratteri*)

Funzioni Semantiche/2

Funzioni Ausiliarie

$\mathtt{N} : \mathtt{VL} \longrightarrow \mathtt{Int}$ (*iniettiva*, *i.e. costruttore* [*suriettiva*??])

$\mathtt{C} : \mathtt{VL} \longrightarrow \mathtt{Char}$ (*iniettiva*, *i.e. costruttore*)

$\mathtt{V} : \mathtt{Int} \cup \mathtt{Char} \longrightarrow \mathtt{VL}$ (*iniettiva*, *i.e. costruttore*)

$\mathtt{NtoVL} : \mathtt{Int} \longrightarrow \mathtt{VL}$

$\mathtt{NtoVL} = \lambda \mathtt{x}.\mathtt{V}(\mathtt{x})$

$\mathtt{CtoVL} : \mathtt{Char} \longrightarrow \mathtt{VL}$

$\mathtt{CtoVL} = \lambda \mathtt{x}.\mathtt{V}(\mathtt{x})$

$\mathtt{VLtoN} : \mathtt{VL} \longrightarrow \mathtt{Int}_{\perp}$

$\mathtt{VLtoN} = \lambda \mathtt{x}.\mathtt{if}((\mathtt{x} \in \mathtt{Int}), \mathtt{N}(u), \perp_{\mathtt{Int}})$

$\mathtt{VLtoC} : \mathtt{VL} \longrightarrow \mathtt{Char}$

$\in \mathtt{Int} : \mathtt{VL} \longrightarrow \mathtt{TruthV}$

$\in \mathtt{Int} = \lambda \mathtt{x}.\mathtt{x}\ eq\ \mathtt{V}(\mathtt{N}(u))$

$\in \mathtt{Char} : \mathtt{VL} \longrightarrow \mathtt{TruthV}$

$\mathtt{MV} : \mathtt{Mem} \longrightarrow \mathtt{Val}$

$\mathtt{IntoVal} : \mathtt{VL} \longrightarrow \mathtt{Val}$

Linguaggi con Blocchi_inline e/o Blocchi Procedura/1

Table5 – Linguaggi Imperativi : Scoping Statico

Funzioni Semantiche

$\mathcal{D}[\![\mathtt{D}]\!]_\rho$: *Env* (*Significato delle Dichiarazioni*)

$\mathcal{D}[\![\mathtt{Proc\ I()\ C}]\!]_\rho = \mathtt{bind}(\mathtt{I}, \mathcal{M}[\![\mathtt{C}]\!]_\rho, \rho)$

$\mathcal{M}[\![\mathtt{C}]\!]_\rho : \mathtt{State} \to \mathtt{State}$ (*Invocazione*)

$\mathcal{M}[\![\mathtt{Call\ I()}]\!]_\rho = \mathtt{find}(\mathtt{I}, \rho)$

Domini Ausiliari

$\mathtt{Den} ::= \mathtt{Loc} + \mathtt{ProcFun}$ (*Unione disgiunta*)

$\mathtt{ProcFun} ::= \mathtt{State} \to \mathtt{State}$ (*Valori Procedure*)

Funzioni Ausiliarie

$\mathtt{Q} : (\mathtt{State} \to \mathtt{State}) \to \mathtt{ProcFun}$ (*iniettiva, i.e. costruttore*)

Blocchi Procedura: Implementazione con Chiusure

$$\mathcal{M}[\![\mathtt{C}]\!]_\rho \quad \textit{vs.} \quad (\mathtt{C}, \rho) = (p_C, p_\rho)$$

- La semantica delle procedure o funzioni con scoping statico, introduce la **chiusura** (Landin 1966) che è una coppia $(\mathtt{C}, \rho)$ contenente un codice $\mathtt{C}$ (oppure $\mathtt{E}$ per funzioni), ed un ambiente.
- **Stack di AR**. Sia AR_{top} l'AR in cui si invoca la procedura I, un nuovo AR_I è creato e posto sul top. AR_I ha:

$$pc = p_C; cd = AR_{top}; cs = p_\rho.$$

Linguaggi con Blocchi_inline e/o Blocchi Procedura/2

Table6 – Linguaggi Imperativi : Scoping Dinamico

Funzioni Semantiche

$\mathcal{D}[\![\texttt{D}]\!]_\rho$: *Env* (*Dichiarazione*)

$\mathcal{D}[\![\texttt{Proc I() C}]\!]_\rho = \texttt{bind}(\texttt{I}, \lambda\delta\,.\,\mathcal{M}[\![\texttt{C}]\!]_\delta)$

$\mathcal{M}[\![\texttt{C}]\!]_\rho$: $\texttt{State} \rightarrow \texttt{State}$ (*Invocazione*)

$\mathcal{M}[\![\texttt{Call I()}]\!]_\rho = \texttt{find}(\texttt{I}, \rho)(\rho)$

Procedure con scoping dinamico: Implementazione

$$\lambda\delta\,.\,\mathcal{M}[\![\mathtt{C}]\!]_\delta \quad \textit{vs.} \quad \textit{Com} = p_C$$

- La semantica delle procedure o funzioni con scoping dinamico ricorre al semplice p_C
- **Stack di AR**. Sia AR_{top} l'AR in cui si invoca la procedura I, un nuovo AR_I è creato e posto sul top. AR_I ha:

$$pc = p_C; cd = AR_{top}.$$

Blocchi: Dichiarazioni Sequenziali, Parallele, Miste/1

Table7 – Linguaggi : Dichiarazione Sequenziale

Domini Sintattici

$\texttt{D} ::= \texttt{Proc I() C;} \mid \texttt{D D} \mid \texttt{Const I} = \texttt{VL} \ldots$

Funzioni Semantiche

$\mathcal{D}[\![\texttt{D}]\!]\rho : \mathit{Env}$

$$\mathcal{D}[\![\texttt{D}_1\ \texttt{D}_2]\!]\rho = \mathcal{D}[\![\texttt{D}_2]\!]_{\mathcal{D}[\![\texttt{D}_1]\!]\rho}$$
$$\mathcal{D}[\![\texttt{Const I} = \texttt{VL}]\!]\rho = \texttt{bind}(\texttt{I}, \texttt{IntoVal}(\texttt{VL}), \rho)$$

Domini Ausiliari

$\texttt{Den} ::= \texttt{Loc} + \texttt{ProcFun} + \texttt{VL}$ (*unione disgiunta*)

Blocchi: Dichiarazioni Sequenziali, Parallele, Miste/2

Table8 – Linguaggi : Dichiarazioni Mutuamente Ricorsive

Domini Sintattici

$$\mathtt{D} ::= \ldots \mid \mathtt{Mut\ D_1\ D_2\ Ally} \mid \ldots$$

Funzioni Semantiche

$$\mathcal{D}[\![\mathtt{D}]\!]\rho : \mathit{Env}$$

$$\mathcal{D}[\![\mathtt{Mut\ D_1\ D_2\ Ally}]\!]\rho = \mathtt{Y}\ \delta .\ (\mathcal{D}[\![\mathtt{D_1}]\!]\delta \circ \mathcal{D}[\![\mathtt{D_2}]\!]\delta \circ \rho)$$
$$\mathit{where} : f \circ\ g = \lambda\, x .\ g(f(x))$$

Blocchi: Dichiarazioni Sequenziali, Parallele, Miste/3

Example

Siano `A` e `B` due identificatori. Consideriamo il seguente frammento di programma:

```
...
{...
  Mut
    Proc A() {Call B();}
    Proc B() {Call A();}
  Ally
...
```

Applichiamo alla dichiarazione la semantica definita per il costrutto `Mut_Ally`.

Example - continua1

$$\mathtt{g} \equiv \mathtt{Y}\,\delta\,.\mathtt{bind}(\mathtt{A}, \mathcal{M}[\![\mathtt{Call\ B()}]\!]_\delta, \delta) \circ \mathtt{bind}(\mathtt{B}, \mathcal{M}[\![\mathtt{Call\ A()}]\!]_\delta, \delta) \circ \rho$$

Calcoliamo le prime tre approssimazioni con il funzionale:

$$\mathtt{H} \equiv \lambda\,\delta\,.\mathtt{bind}(\mathtt{A}, \mathcal{M}[\![\mathtt{Call\ B()}]\!]_\delta, \delta) \circ \mathtt{bind}(\mathtt{B}, \mathcal{M}[\![\mathtt{Call\ A()}]\!]_\delta, \delta) \circ \rho$$

otteniamo:

$$\mathtt{H}^0_\perp \equiv \perp$$

Example - continua2

$$\mathtt{H} \equiv \lambda\,\delta\,.\mathtt{bind}(\mathtt{A}, \mathcal{M}[\![\mathtt{Call\ B()}]\!]_\delta, \delta) \circ \mathtt{bind}(\mathtt{B}, \mathcal{M}[\![\mathtt{Call\ A()}]\!]_\delta, \delta) \circ \rho$$

otteniamo:

$$\mathtt{H}^0_\perp \equiv \perp$$
$$\mathtt{H}^1_\perp \equiv \mathtt{bind}(\mathtt{A}, \mathcal{M}[\![\mathtt{Call\ B()}]\!]_\perp, \perp) \circ \mathtt{bind}(\mathtt{B}, \mathcal{M}[\![\mathtt{Call\ A()}]\!]_\perp, \perp) \circ \rho$$
$$\mathtt{H}^2_\perp \equiv$$
$$\mathtt{bind}(\mathtt{A}, \mathcal{M}[\![\mathtt{Call\ B()}]\!]_{\mathtt{H}^1_\perp}, \mathtt{H}^1_\perp) \circ \mathtt{bind}(\mathtt{B}, \mathcal{M}[\![\mathtt{Call\ A()}]\!]_{\mathtt{H}^1_\perp}, \mathtt{H}^1_\perp) \circ \rho$$
$$\mathtt{H}^3_\perp \equiv$$
$$\mathtt{bind}(\mathtt{A}, \mathcal{M}[\![\mathtt{Call\ B()}]\!]_{\mathtt{H}^2_\perp}, \mathtt{H}^2_\perp) \circ \mathtt{bind}(\mathtt{B}, \mathcal{M}[\![\mathtt{Call\ A()}]\!]_{\mathtt{H}^2_\perp}, \mathtt{H}^2_\perp) \circ \rho$$

$$\mathtt{g} = \mathtt{H}^\omega_\perp$$

Lo si verifichi applicandolo opportunamente...

Blocchi: Dichiarazioni Sequenziali, Parallele, Miste/3

Example

Siano `A` e `B` due identificatori. Consideriamo il seguente frammento di programma: ...

```
{...
  Const int x = 0;
  Mut
    Fun int A() {if (x=0) then Call B(); else return 5}
    Fun int B() {if (x!=0) then Call A(); else return 7}
  Ally
...
```

Mostrare che $g = H^2_\perp$

Dichiarazioni Sequenziali, Parallele, Miste/3: Uso e Implementazione

- **Uso:** Le dichiarazioni (di tutte e tre le forme) sono il meccanismo principale e *trasparente* per il Naming nei programmi
  - sequenziale. Ordinamento totale delle dipendenze (riferimenti) tra dichiarazioni
  - parallelo. Definizioni mutuamente riferenti (procedure mutuamente ricorsive, classi mutuamente riferenti)
  - mista. Combinazione delle due esplicita (costrutto mutually) o implicita (per tipologie di dichiarazioni: sequenziale per variabili, parallela per procedure)
- **Implementazione:** Basata sulle strutture Env ed integrata con analizzatori di semantica statica.

Dichiarazioni Sequenziali, Parallele, Miste/4: Implementazione

- **Implementazione:** Basata sulle strutture Env ed integrata con analizzatori di semantica statica che controllano staticamente la *corretteza* (vedi esempio lucido seguente) della composizione di dichiarazioni.
  - Costruzione di un Env E ad ogni dichiarazione (possibilmente composta, ovvero D D):
  - inserimento dei bindings nell'ordine in cui sono incontrati nel testo
  - Procedure hanno chiusura con pointer di ambiente E.
- Un unica implementazione per le tre forme. La correttezza dell'implementazione è affidata alla corretettezza dell'analisi statica.

Dichiarazioni Sequenziali, Parallele, Miste/5: Analisi Statica

- Analisi Statica deve rilevare uso di dichiarazioni la cui semantica conduce a definizioni circolari

Example

Si mostri che la semantica di mutually è definita quando applicata al seguente frammento. ...

```
{...
  Mut
     int x=  y}
     int y=  x}
  Ally
...
```

Si commentino i problemi ai quali questo fatto puó condurre (si pensi anche anche alla dichiarazione delle procedure data precedentemente).