

Capitolo 1

Induzione strutturale

Introduzione

Sia A un insieme, sia \triangleleft una relazione binaria definita su A ($\triangleleft : A \times A$).

Def. \triangleleft è b.f. (ben fondata) se $\nexists \dots \triangleleft a_i \triangleleft \dots \triangleleft a_1 \triangleleft a_0$
ovvero non esistono catene infinite discendenti.
(A, \triangleleft) è un insieme ben fondato se \triangleleft è b.f.

Sia \trianglelefteq la chiusura riflessiva e transitiva di \triangleleft b.f. su A .

Lemma 1. \trianglelefteq non è mai b.f.

Dimostrazione. Sia $a_i \in A$. $\exists \dots a_i \trianglelefteq a_i \trianglelefteq \dots$ □

Def. Sia $a \in A$. a è *minimale* in A rispetto a \triangleleft se $\forall b \triangleleft a, b \notin A$.

Lemma 2. \triangleleft è ben fondato su A se e solo se ogni $B \subseteq A$ ha un elemento minimale rispetto a \triangleleft .

Dimostrazione. • \Rightarrow) Da $B \subseteq A$ e (A, \triangleleft) b.f. si ha che non esiste $\dots \triangleleft b_i \triangleleft \dots \triangleleft b_0$, quindi $\exists b_n$ t.c. $b_n \triangleleft \dots \triangleleft b_i \triangleleft \dots \triangleleft b_0$ ovvero b_n è minimale in B rispetto a \triangleleft .

• \Leftarrow) Per assurdo. Supponiamo che esista $\dots \triangleleft a_i \triangleleft \dots \triangleleft a_0$, ovvero $\neg(\triangleleft \text{ b.f. })$. Allora l'insieme $B = \{a_i | i \in \mathbb{N}\}$ (insieme degli elementi della sequenza) non ha un minimale, cosa che contraddice l'ipotesi. Quindi una tale sequenza non esiste, ovvero \triangleleft b.f. □

1.1 Principio di induzione noetheriana

Teorema 1 (Principio di induzione noetheriana (prima forma)). *Sia \mathcal{P} una proprietà su (A, \triangleleft) b.f. .*

$$\forall a \in A. \mathcal{P}(a) \Leftrightarrow \forall a \in A. ([\forall b \triangleleft a. \mathcal{P}(b)] \Rightarrow \mathcal{P}(a))$$

Dimostrazione. • \Rightarrow) Ovvio.

- \Leftarrow) Per assurdo.
Supponiamo

$$\forall a \in A. ([\forall b \triangleleft a. \mathcal{P}(b)] \Leftarrow \mathcal{P}(a)) \quad (1.1)$$

e

$$\exists c \in A. \neg \mathcal{P}(c) \quad (1.2)$$

Sia $C = \{c \in A \mid \neg \mathcal{P}(c)\} \subseteq A$.

Per il lemma 2, $\exists \hat{c}$ minimale di C rispetto a \triangleleft , allora $\neg \mathcal{P}(\hat{c})$

Per \hat{c} minimale, $\forall b \triangleleft \hat{c}. b \notin C$, allora $\mathcal{P}(b)$, ovvero per (1.1) $\mathcal{P}(\hat{c})$

□

Def. $\text{base}_A = \{a \in A \mid a \text{ è minimale in } A \text{ rispetto a } \triangleleft\}$.

Osserviamo che $\forall a \in \text{base}_A, \forall b \in A. b \not\triangleleft a$.

Teorema 2 (Principio di induzione noetheriana - seconda forma). *Sia \mathcal{P} una proprietà su (A, \triangleleft) b.f. .*

$$\forall a \in A. \mathcal{P}(a) \Leftrightarrow \left(\begin{array}{c} \forall a \in \text{base}_A. \mathcal{P}(a) \\ \wedge \\ \forall a \in (A \setminus \text{base}_A). ([\forall b \triangleleft a. \mathcal{P}(b)] \Rightarrow \mathcal{P}(a)) \end{array} \right)$$

Teorema 3 (Induzione matematica). *Sia $A = \mathbb{N}$.*

Sia $n \triangleleft m \Leftrightarrow m = n + 1 \quad n, m \in \mathbb{N}$.

Osserviamo che \triangleleft è ben fondata: $0 \triangleleft 1 \triangleleft 2 \triangleleft \dots$

Osserviamo $\text{base}_{\mathbb{N}} = \{0\}$.

$$\forall m \in \mathbb{N}. \mathcal{P}(m) \Leftrightarrow \left(\begin{array}{c} \mathcal{P}(0) \\ \wedge \\ \forall m > 0. (\mathcal{P}(m-1) \Rightarrow \mathcal{P}(m)) \end{array} \right)$$

Def. $a^\triangleleft = \{b \in A \mid b \triangleleft a\}$ con (A, \triangleleft) b.f. .

Def. Sia $f : A \rightarrow B$, sia $A' \subseteq A$. $f|_{A'} = \{(a, f(a)) \mid a \in A'\}$.

Teorema 4 (di ricorsione / delle definizioni noetheriane). *Sia (A, \triangleleft) b.f.*

$$\forall a \in A, \forall h : a^\triangleleft \rightarrow B. F(a, h) \in B$$

(F è detta operatore di composizione). Allora

$$\exists! f : A \rightarrow B \text{ t.c. } \forall a \in A. f(a) = F(a, f|_{a^\triangleleft})$$

Esempio. Sia $A = \mathbb{N}$, sia $n \triangleleft m \Leftrightarrow m = n + 1$.

$$\text{Fact}(n) = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot \text{Fact}(n-1) & \text{se } n > 0 \end{cases}$$

In questo caso, $f = \text{Fact}$ e F è la moltiplicazione.

$$f(n) = F(n, f|_{n^\triangleleft}) = n \cdot f(n-1)$$

Teorema 5 (di induzione sulle grammatiche libere dal contesto). *Dato:*

$$G = \{N, \Sigma, P, S\}$$

$$\triangleleft \subseteq (N \cup \Sigma) \times (N \cup \Sigma)^+$$

$$\forall A = x_1, \dots, x_n \in \mathcal{P}, x_1 \triangleleft x_2 \triangleleft \dots \triangleleft x_n$$

Dall'ultima possiamo osservare che la lunghezza della stringa più piccola è minore della lunghezza dell'intera stringa, quindi non posso creare catene infinitamente discendenti, per questo la relazione sulle stringhe di un linguaggio si può considerare ben fondata.

Esempio.

$$B = 0|1|B0|B1$$

$$0 \triangleleft 1 \triangleleft B0 \triangleleft B1$$

Osserviamo che abbiamo espresso un ordine per gli elementi 0, 1, quest'ordine per semplicità lo possiamo prenderla dalla semantica induttiva sui numeri naturali.

Teorema 6 (Principio d'induzione sulle grammatiche libere dal contesto).

$$\forall \omega \in L(G). \mathcal{P}(\omega) \Leftrightarrow$$

$$\forall A = x_1, \dots, x_n \in P \left(\left[\bigwedge_{i=1}^n \forall \omega \in L(x_i). \mathcal{P}(\omega) \right] \Rightarrow \forall \omega \in L(x_1, \dots, x_n). \mathcal{P}(\omega) \right)$$

Per ogni produzione, le stringhe generate da ogni simbolo del lato sinistro soddisfano \mathcal{P} implica che le stringhe generate dall'intera produzione soddisfano \mathcal{P} , questo è equivalente a dire che \mathcal{P} vale per ogni stringa appartenente al linguaggio.

Esempio. Cerco di dimostrare una proprietà \mathcal{P} su $B = 0|1|B0|B1$

$$\forall \omega \in L(B); \omega = \omega 0$$

$$\omega' \in L(B) \Rightarrow \omega \text{ è un numero pari}$$

Soluzione:

Riscrittura principio d'induzione prima in maniera compatta poi in maniera esplicita

$$\left(\left[\bigwedge_{i=1}^n \forall \omega \in L(x_i). \mathcal{P}(\omega) \right] \Rightarrow \forall \omega \in L(B0). \mathcal{P}(\omega) \right)$$

$$(\forall \omega \in L(B). \mathcal{P}(\omega)) \Rightarrow \forall \omega \in L(B0). \mathcal{P}(\omega)$$

Se riesco a dimostrare che la proprietà è vera per tutte le stringhe formate da B , allora posso assumere che per tutte le stringhe che conterranno $B0$ la proprietà sarà vera. (0 è un caso base e lo diamo per vero)

Capitolo 2

Definizione di Linguaggio

TODO

Composizionalità: la proprietà per cui ciascuna stringa deve essere funzione solo dei componenti della stringa stessa.

Modularità: la proprietà per cui, aggiungendo dei costrutti ad un linguaggio L , non devo ridefinire la semantica dello stesso.

- posso quindi definire un linguaggio in maniera incrementale, ovvero, partendo da un nucleo centrale, posso aggiungere dello zucchero sintattico senza modificare il nucleo di partenza.

Capitolo 3

Semantica

- **Operazionale:** COME eseguire un programma \mathcal{P} .
- **Denotazionale:** CHE COSA si ottiene dall'esecuzione di \mathcal{P} .
- **Assiomatica:** verifica SE il programma \mathcal{P} è corretto rispetto ad una data proprietà.

Def. dato un programma \mathcal{P} :

$$\dots z := 2; y := z; y := y + 1; z := y \dots \quad (3.1)$$

il **supporto a tempo di esecuzione** è definito come:

$$[z = 0, y = 0] \equiv \rho(z) = 0, \rho(y) = 0 \quad (3.2)$$

TODO

3.1 Definizione formale

Def. Una semantica è una quadrupla:

$$(L, M, \varepsilon_{i \in I}, \equiv_m) \quad (3.3)$$

dove:

- L è il linguaggio oggetto. $l \in L$ sono le stringhe del linguaggio a cui bisogna assegnare un significato.
- M è il meta linguaggio, che definisce gli oggetti che si usano per assegnare la semantica (o il significato) ad $L \Rightarrow \forall l \in L \exists m \in M | m(l) = \text{significato}$.
- $\varepsilon_{i \in I}$ sono funzioni di interpretazione semantica $\Rightarrow \varepsilon_{i \in I} : L \rightarrow M$.
- \equiv_m è l'equivalenza semantica $\subseteq M \times M$.

Lemma 3.

$$l, l' \in L, l \equiv_l l' \iff \varepsilon[l] \equiv_m \varepsilon[l'] \quad (3.4)$$

Dimostrazione. L'equivalenza sul linguaggio esiste solo e soltanto se gli elementi del meta linguaggio ($\varepsilon[l] \in M$) sono equivalenti. \square

Lemma 4.

$$\equiv_M = \text{id} \not\Rightarrow \equiv_L = \text{id} \quad (3.5)$$

Dimostrazione. Se l'equivalenza su M è l'identità cioè non implica la stessa proprietà per l'equivalenza sul linguaggio L (?) \square

3.2 Semantica operativa

Def. Un **sistema di transizione** T_G è un grafo t.c.

$$T_G = (\mathcal{P} \times \rho, \longrightarrow, I, F)$$

dove

- \mathcal{P} sono i programmi (ad es. assegnazioni di (3.1))
- ρ sono gli ambienti del supporto a tempo di esecuzione (ad es. (3.2))
- $\longrightarrow \subseteq (\mathcal{P}, \rho) \times (\mathcal{P}, \rho)$
- $I \subseteq (\mathcal{P}, \rho)$ iniziali
- $F \subseteq (\mathcal{P}, \rho)$ finali

Def. La **semantica operativa** è una funzione

$$\mathcal{E}_{op} : L(G) \times \rho_{init} \longrightarrow T_G$$

utilizzando la definizione formale (3.3) definiamo:

- L : linguaggio oggetto (che definiamo)
- M : $\langle S, \longrightarrow \rangle$ dove $S = \{ \langle l, \rho : \text{Var}(l) \longrightarrow \text{Val} \rangle \mid l \in L \}$ ovvero tutti i possibili stati di tutti i possibili programmi generati da L , e $\longrightarrow \subseteq S \times S$
- $\varepsilon_{i \in I} \equiv$ definizione operativa data a inizio paragrafo.
- \equiv_M è l'identità sui grafi (isomorfismo)

3.3 Semantica denotazionale

Def. La **semantica denotazionale** è definita come

$$\mathcal{E}_{den} : L(G) \times \rho_{init} \longrightarrow \rho_{fin}$$

utilizzando la definizione formale (3.3):

- L è il linguaggio che definiamo
- $M = \{ \rho : \text{Var}(l) \longrightarrow \text{Val} \mid l \in L \}$ ovvero l'insieme delle funzioni che danno valore alle variabili di L .
- $\varepsilon_{i \in I} : L \longrightarrow M, l \longrightarrow \text{val}$
- \equiv_m è l'identità

Rappresenta solo il risultato dell'esecuzione del programma, ovvero l'insieme degli ambienti finali.

3.4 Semantica assiomatica

Def. Una **regola di inferenza** stabilisce che:

$$\frac{p_1 \wedge \dots \wedge p_n}{C}$$

se $p_1 \wedge \dots \wedge p_n$ sono vere allora C è vera.

Def. Ho un **assioma** se:

$$\frac{}{C}$$

ovvero se non ho proprietà su cui basare la conclusione.

Def. La **semantica assiomatica** è definita sulla base di regole di inferenza ed assiomi, e dimostra la veridicità di un programma costruendo un albero le cui foglie sono tutte **assiomi**.

Utilizzando la definizione formale di semantica a (3.3):

- L è il linguaggio che definiamo
- M è l'insieme di alberi di derivazione per $l \in L$
- $\varepsilon_{i \in I}$ sono le interpretazioni degli assiomi e delle regole di inferenza
- \equiv_M è l'identità sugli alberi di derivazione

3.5 Contesto

Def. Un **contesto** $C[\cdot]$ per un programma è un programma con un buco (una parte non specificata).

1. **Principio di Substitutivity:**

$l, l' \in L$ che possono comparire nello stesso contesto $C[\cdot]$:

$$l \equiv_L l' \Rightarrow \forall C[\cdot], C[l] \equiv_L C[l']$$

2. **Principio di Full abstraction:**

$$l, l' \in L, \forall C[\cdot], C[l] \equiv_L C[l'] \implies l \equiv_L l'$$

Capitolo 4

Linguaggio imperativo: IMP

Def. La memoria è rappresentata da una funzione:

$$\sigma : \bigcup_{\tau \in \text{STyp}} \text{Loc}_\tau \rightarrow \text{SVal} \cup \{?, \perp\} \quad (4.1)$$

Dove $?$ rappresenta una locazione di memoria allocata e non inizializzata, mentre \perp rappresenta una locazione non allocata.

Si suppone per comodità che $\forall \tau. |\text{Loc}_\tau| = \infty$, ovvero che la memoria sia infinita.

Def. Si definisce l'estensione di σ (definita su L) a σ_0 (definita su L_0)

$$\sigma[\sigma_0](l) = \begin{cases} \sigma_0(l) & \text{se } l \in L_0 \\ \sigma(l) & \text{se } l \in (L \setminus L_0) \end{cases} \quad (4.2)$$

TODO: ambienti statici e dinamici e loro estensioni, compatibilità di amb.

4.1 Espressioni

Il linguaggio oggetto l delle espressioni è generato dalla grammatica

$$E ::= k \mid \text{id} \mid E_0 \text{ bop } E_1 \mid \text{uop } E$$

4.1.1 Semantica statica

$$\emptyset \vdash_I n : \text{int} \quad (4.3)$$

$$\emptyset \vdash (tt|ff) : \text{bool} \quad (4.4)$$

$$\Delta \vdash \text{id} : \Delta(\text{id}) \quad (4.5)$$

$$\frac{\Delta \vdash E_0 : \tau_0, \Delta \vdash E_1 : \tau_1}{\Delta \vdash E_0 \text{ bop } E_1 : \tau_{\text{bop}}(\tau_0, \tau_1)} \quad (4.6)$$

$$\frac{\Delta \vdash E : \tau}{\Delta \vdash \text{uop } E : \tau_{\text{uop}}(\tau)} \quad (4.7)$$

4.1.2 Semantica dinamica

Il metalinguaggio m è un grafo (sistema di transizione)

$$(< \mathcal{P}, \sigma >, \rightarrow_e \subseteq < \mathcal{P}, \sigma > \times < \mathcal{P}, \sigma >)$$

Si procede per induzione sulla struttura della sintassi.

I **casi base** sono k e id . k è di per sé una configurazione finale. Per id si ha l'assioma:

$$\overline{\rho \vdash < \text{id}, \sigma > \rightarrow < k, \sigma >, k = \sigma(\text{id})} \quad (4.8)$$

Per $E_0 \text{ bop } E_1$ le ipotesi induttive sono E_0, E_1 e si hanno le regole:

$$\frac{\rho \vdash < E_0, \sigma > \rightarrow < E'_0, \sigma >}{\rho \vdash < E_0 \text{ bop } E_1, \sigma > \rightarrow < E'_0 \text{ bop } E_1, \sigma >} \quad (4.9)$$

$$\frac{\rho \vdash < E_1, \sigma > \rightarrow < E'_1, \sigma >}{\rho \vdash < k_0 \text{ bop } E_1, \sigma > \rightarrow < k_0 \text{ bop } E'_1, \sigma >} \quad (4.10)$$

$$\overline{\rho \vdash < k_0 \text{ bop } k_1, \sigma > \rightarrow < k', \sigma >} \quad (4.11)$$

dove k' è il risultato di $k_0 \text{ bop } k_1$.

Per $\text{uop } E$, l'ipotesi induttiva è E e si hanno le regole:

$$\frac{\rho \vdash < E_0, \sigma > \rightarrow < E'_0, \sigma >}{\rho \vdash < \text{uop } E_0, \sigma > \rightarrow < \text{uop } E'_0, \sigma >} \quad (4.12)$$

$$\overline{\rho \vdash < \text{uop } k, \sigma > \rightarrow < k', \sigma >} \quad (4.13)$$

dove k' è il risultato di $\text{uop } k_0$.

TODO: esempio, nota sull'ordine di interpretazione

4.2 Bound Identifiers, Free Identifiers

Un nome, all'interno di un programma, può apparire in posizione:

di definizione quando è definito

legata quando appare dopo essere stato definito, cioè si può determinare la definizione corrispondente

libera quando appare senza essere stato definito

I nomi in posizione di definizione e legata sono detti *bound identifiers*.

I nomi in posizione libera sono detti *free identifiers*.

Def. Si definisce la funzione

$$\text{FI} : \text{IDE} \rightarrow \text{IDE}$$

che restituisce l'insieme dei free identifiers, e la funzione

$$\text{BI} : \text{IDE} \rightarrow \text{IDE}$$

che restituisce l'insieme dei bound identifiers. Vale:

$$\forall \text{prg}. \text{FI}(\text{prg}) \cap \text{BI}(\text{prg}) = \emptyset$$

Lemma 5. *Se in un'espressione sono presenti identificatori liberi non già definiti altrove, l'espressione non può essere valutata.*

$$\text{FI}(E) \not\subseteq I \Rightarrow \Delta \not\vdash_I E : \tau$$

con $I = \{\text{id} \mid \exists(\text{id}, \tau) \in \Delta\}$.

Dimostrazione. La premessa è equivalente ad affermare che

$$\exists x. x \in \text{FI}(E) \wedge x \notin I$$

Si dimostra per induzione sulla struttura della sintassi.

Per il caso base k , si ha $\text{FI}(k) = \emptyset \subseteq I \forall I$.

Per il caso base id , si ha $\text{FI}(\text{id}) = \{\text{id}\}, \text{id} \notin I$. Da ciò $\Delta(\text{id}) = \perp \Rightarrow \Delta \not\vdash_I \text{id} : \tau$.

Per il passo induttivo $E_0 \text{ bop } E_1$, si ha che

$$\exists x \in \text{FI}(E_0 \text{ bop } E_1) \wedge x \notin I$$

o anche, per la definizione di FI

$$(\exists x \in \text{FI}(E_0) \wedge x \notin I) \vee (\exists x \in \text{FI}(E_1) \wedge x \notin I)$$

Nel primo caso, ciò equivale a dire che $\text{FI}(E_0) \not\subseteq I \xRightarrow{\text{HP IND}} \Delta \not\vdash_I E_0 : \tau_0 \Rightarrow$ non vale la 4.6. Il secondo caso è simmetrico.

Per il passo induttivo $\text{uop } E$, si ha che

$$\exists x \in \text{FI}(\text{uop } E)$$

o anche, per la definizione di FI

$$\exists x \in \text{FI}(E) \wedge x \notin I \xRightarrow{\text{HP IND}} \Delta \not\vdash_I E : \tau$$

quindi non vale la 4.7. □

4.3 Dichiarazioni

Il linguaggio l delle dichiarazioni è quello generato dalla grammatica

$$D ::= \text{nil} \mid \text{const } x : \tau = E \mid \text{var } x : \tau = E \mid D_0; D_1 \mid D_0 \text{in} D_1 \mid D_0 \text{and} D_1$$

La funzione ρ viene resa più efficiente per le costanti, trasformandola in

$$\rho : \text{IDE} \rightarrow \text{DVal} \cup \{?, \perp\}, \text{DVal} = \text{Loc} \mid k$$

$$\rho(x) = k \Rightarrow x \text{ è costante}$$

Per verificare la compatibilità $\Delta : \rho$ bisogna estendere DTyp , aggiungendo per ogni τ un tipo corrispondente $\tau \text{ loc}$, che indica una *locazione di tipo* τ .

$$\Delta \vdash E : \tau, \text{var } x : \tau = E \rightsquigarrow \Delta \vdash x : \tau \text{ loc}$$

4.3.1 Free Identifiers

$$\text{FI}(\text{nil}) = \emptyset \quad (4.14)$$

$$\text{FI}(\text{const } x : \tau = E) = \text{FI}(E) \quad (4.15)$$

$$\text{FI}(\text{var } x : \tau = E) = \text{FI}(E) \quad (4.16)$$

$$\text{FI}(D_0; D_1) = \text{FI}(D_0) \cup (\text{FI}(D_1) \setminus \text{BI}(D_0)) \quad (4.17)$$

$$\text{FI}(D_0 \text{in} D_1) = \text{FI}(D_0) \cup (\text{FI}(D_1) \setminus \text{BI}(D_0)) \quad (4.18)$$

$$\text{FI}(D_0 \text{and} D_1) = \text{FI}(D_0) \cup \text{FI}(D_1) \quad (4.19)$$

4.3.2 Semantica statica

$$\frac{}{\Delta \vdash \text{nil} : \Delta} \quad (4.20)$$

$$\frac{\Delta \vdash_I E : \tau}{\Delta \vdash \text{const } x : \tau = E : [x : \tau]} \quad (4.21)$$

$$\frac{\Delta \vdash_I E : \tau}{\Delta \vdash \text{var } x : \tau = E : [x : \tau \text{ loc}]} \quad (4.22)$$

$$\frac{\Delta \vdash D_0 : \Delta_0, \Delta[\Delta_0] \vdash D_1 : \Delta_1}{\Delta \vdash D_0; D_1 : \Delta_0[\Delta_1]} \quad (4.23)$$

$$\frac{\Delta \vdash D_0 : \Delta_0, \Delta[\Delta_0] \vdash D_1 : \Delta_1}{\Delta \vdash D_0 \text{in} D_1 : \Delta_1} \quad (4.24)$$

$$\frac{\Delta \vdash D_0 : \Delta_0, \Delta \vdash D_1 : \Delta_1}{\Delta \vdash D_0 \text{and} D_1 : \Delta_0, \Delta_1} \quad (4.25)$$

Nell'ultimo caso, si ha la condizione che gli insiemi degli identificatori di D_0 e D_1 siano distinti. Si considera inoltre $\Delta_0, \Delta_1 = \Delta_0[\Delta_1] = \Delta_1[\Delta_0]$.