aglaia norza

Linguaggi di Programmazione

appunti delle lezioni
libro del corso: non usato, integrati con le dispense del professor Cenciarelli

21/10/2025

thisisaglaia@gmail.com
github.com/AglaiaNorza

Contents

1. Algebre induttive

1.1. I numeri naturali

def. 1: Assiomi di Peano

L'insieme $\mathbb{N}$ dei numeri naturali si può definire mediante i cinque **assiomi di Peano**:

1. $0 \in \mathbb{N}$
2. $n \in \mathbb{N} \Rightarrow \text{succ}(n) \in \mathbb{N}$
3. $\nexists n \in \mathbb{N} \mid 0 = \text{succ}(n)$
4. $\forall n, m \text{ succ}(n) = \text{succ}(m) \Rightarrow n = m$ (iniettività)
5. $\forall S \subseteq \mathbb{N}\ (0 \in S \wedge (n \in S \Rightarrow \text{succ}(n) \in S) \Rightarrow S = \mathbb{N})$ (assioma di induzione)

assioma di induzione

L'assioma di induzione è necessario per evitare di equiparare ai numeri naturali insiemi che, essenzialmente, contengono una struttura come quella di $\mathbb{N}$, e un "qualcosa in più". (Se all'interno dell'insieme A che stiamo considerando esiste un altro sottoinsieme proprio che rispetta gli altri assiomi, A non rispetterà il quinto assioma di Peano).

In più, il quinto assioma di Peano ci fornisce essenzialmente una definizione insiemistica di induzione.

def. 2: Principio di Induzione

L'induzione può essere definita, basandosi sulle "proprietà" invece che sull' insiemistica, come segue:

$$\forall P \quad \frac{P(0), \quad P(n) \Rightarrow P(n+1)}{\forall n\ P(n)}$$

(la notazione equivale a $P(0) \wedge (P(n) \Rightarrow P(n+1)) \Rightarrow \forall n P(n)$)

Possiamo dimostrare che il quinto assioma di Peano è equivalente al principio di induzione (in quanto i concetti di "proprietà" e "sottoinsieme" sono equivalenti).

Infatti, ad ogni proprietà corrisponde un sottoinsieme i cui elementi sono esattamente quelli che soddisfano tale proprietà

Prendiamo quindi $S = \{n \in \mathbb{N} \mid P(n) \text{ è vera}\}$.

In questo modo, dire $P(0)$ equivale a dire $0 \in S$, e dire $P(n) \Rightarrow P(n+1)$ equivale a dire $n \in S \Rightarrow n+1 \in S$. E, allo stesso modo, dire $\forall n P(n)$ equivale a dire $\forall n, n \in S$, ovvero $S = \mathbb{N}$.

def. 3: Numeri di von Neumann

Un altro modo di descrivere i numeri naturali viene dal matematico **John von Neumann**, che definisce i numeri naturali ("numeri di von Neumann", $\mathcal{N}$) in questo modo:

- $0_{\mathcal{N}} = \emptyset$ (ovvero $\{\}$)
- $1_{\mathcal{N}} = \{0_{\mathcal{N}}\}$ (ovvero $\{\{\}\}$)
- $2_{\mathcal{N}} = \{0_{\mathcal{N}}, 1_{\mathcal{N}}\}$ (ovvero $\{\ \{\}, \{\{\}\}\ \}$)
- ...

I numeri di von Neumann rispettano gli assiomi di peano! (dalle dispense)

1.2. Algebre, algebre induttive

nota: insieme unità e funzione nullaria

Ci è utile definire l'**insieme unità** $\mathbb{1} = \{*\}$. $\mathbb{1}$ è un insieme formato da un solo elemento (non ci interessa quale).

Un altro concetto che ci servirà è quello di **funzione costante** o **nullaria**. Una funzione nullaria f è tale che:

$$f : \mathbb{1} \to A \mid f() = a \quad a \in A$$

(chiaramente, essa è sempre iniettiva).

nota

Una funzione nullaria su un insieme A può essere vista come un elemento di A (un qualsiasi insieme A è isomorfo a all'insieme di funzioni $\mathbb{1} \to A$ (l'insieme di funzioni $\mathbb{1} \to A$ ha la stessa cardinalità di A), il che ci permette di **trattare gli elementi di un insieme come funzioni**.

def. 4: Algebra

Una **algebra** è una tupla (A, Γ), dove:

- A è l'insieme di riferimento ("carrier" o "insieme sottostante")
- $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_i\}$, è l'insieme di funzioni chiamate "operazioni fondamentali" o "costruttori" dell'algebra

  la segnatura dei costruttori è: $\gamma_i : A^{\alpha_i} \times K_i \to A$.

nota

Tra le algebre, consideriamo anche le algebre eterogenee, che prendono argomenti da insiemi diversi da A.

def. 5: Chiusura di un insieme rispetto ad un'operazione

Sia $f : A^n \times K \to A$ un'operazione su A con parametri esterni $K = (K_1 \times \cdots \times K_m)$.

Un insieme $S \subseteq A$ si dice **chiuso** rispetto ad f quando:

$$a_1, \ldots, a_n \in S \;\Rightarrow\; f(a_1, \ldots, a_n, k_1, \ldots, k_n) \in S$$

nota!

Data un'operazione f che prende solo elementi esterni all'insieme S (come per esempio la funzione nullaria $\mathbb{1} \to A$), un insieme S si dice chiuso rispetto a $f \iff \mathrm{Im}(f) \subseteq S$).

def. 6: Algebra induttiva

Un'algebra A, Γ si dice **induttiva** quando:

1. tutte le $\gamma_i \in \Gamma$ sono iniettive
2. $\forall i, j \mid i \neq j,\ \mathrm{Im}(\gamma_i) \cap \mathrm{Im}(\gamma_j) = \emptyset$, ovvero tutte le γ_i hanno immagini disgiunte
3. $\forall S \subseteq A$, se S è chiuso rispetto a tutte le γ_i, allora $S = A$ (ovvero il principio di induzione è rispettato)

terza condizione

La terza condizione pone quindi che A sia la più piccola sotto-algebra di se stessa (ovvero non abbia sotto-algebre diverse da se stessa).

nota

Le tre condizioni garantiscono quindi che:

- ci sia solo un modo per costruire ogni elemento dell'algebra (*i, ii*)
- non ci siano "elementi inutili" (*iii*)

Vediamo come possiamo costruire $\mathbb{N}$ come algebra induttiva.

La definizione di algebra induttiva non considera il concetto di "elemento", quindi, per il primo assioma di Peano, usiamo una *funzione costante* $\mathbb{0}$, con segnatura:

$$\mathbb{1} \times \mathbb{N} : x \to 0$$

Abbiamo quindi una tupla $(\mathbb{N}, \{\mathrm{succ}, \mathbb{0}\})$.

Per dimostrare che questa tupla sia un'algebra induttiva, dobbiamo ora verificare le tre condizioni:

1. tutte le γ_i sono induttive:
   - $\mathbb{0}$ è necessariamente induttiva
   - succ è induttiva per il secondo assioma di Peano
2. tutti i costruttori hanno immagini disgiunte:
   - grazie al terzo assioma di Peano ($\nexists n \in \mathbb{N} \mid 0 = \mathrm{succ}(n)$), sappiamo che succ e $\mathbb{0}$ hanno immagini disgiunte

3. principio di induzione:

    • è verificato dal quinto assioma di Peano ($0 \in S$ corrisponde alla chiusura rispetto a $\mathbb{0}$ e $n \in \mathbb{N} \Rightarrow \text{succ}(n) \in \mathbb{N}$ corrisponde alla chiusura rispetto a succ)

alberi binari come algebre induttive

L'insieme degli alberi binari finiti ($\texttt{B-trees}$, leaf, branch), dove:

- $\texttt{B-trees} = \{t \mid t$ è una foglia, oppure $t = \langle t_1, t_2 \rangle$ con $t_1, t_2 \in \texttt{B-trees}\}$
- leaf: $1 \to \texttt{B-trees}$ (foglia)
- branch: $\texttt{B-trees} \times \texttt{B-trees} \to \texttt{B-trees} : (t_{sx}, t_{dx}) \to t$ (costruisce rami in modo che t_{sx} e t_{dx} siano i due sottoalberi di t)

è un'algebra induttiva.

theorem 1: numero di nodi di un albero binario

Un albero binario con n foglie ha $2n - 1$ nodi

proof!

Si può dimostrare per induzione strutturale sui costruttori degli alberi.

- (**caso base**): la proprietà è vera per l'albero formato da una sola foglia costruito con leaf ($\circ$) - esso ha infatti $n = 1$ foglie e $2n - 1 = 1$ nodi.
- (**ipotesi induttiva**): ogni argomento dei costruttori rispetta la proprietà
- dobbiamo quindi verificare che il costruttore branch, dati due argomenti che rispettano la proprietà, la mantenga
- (**passo induttivo**): abbiamo $t = \text{branch}(t_1, t_2)$.

  Sia $n = n_1 + n_2$ il numero di foglie di t, dove le foglie di t_1 sono n_1 e quelle di t_2 sono n_2.

  Per ipotesi, t_1 ha $2n_1 - 1$ nodi e t_2 ne ha $2n_2 - 1$. Dunque, t avrà $(2n_1 - 1) + (2n_2 - 1) + 1$ nodi, ovvero $2(n_1 + n_2) - 1 = 2n - 1$, qed.

liste finite come algebra induttiva

Dato un insieme A, indichiamo con $A - list$ l'insieme delle liste finite di elementi di A.

La tupla $(A - list,$ empty, cons) è un'algebra induttiva, dove:

- empty: $\mathbb{1} \to A - list$ è la funzione costante che restituisce la **lista vuota** "$\langle\rangle$".
- cons: $A \times A - list \to A - list : \text{cons}(3, \langle 5, 7 \rangle) = \langle 3, 5, 7 \rangle$ è la funzione che **costruisce una lista** aggiungendo un elemento in testa

Si tratta di un'algebra induttiva (notiamo che i due costruttori hanno immagini chiaramente disgiunte, sono entrambi chiusi per $A - list$, e c'è un unico modo per costruire ogni lista).

liste infinite

Le liste infinite non possono essere un'algebra induttiva, in quanto contengono una sotto-algebra induttiva (quella delle liste finite).

i booleani come algebra non induttiva

Consideriamo l'algebra (B, not), dove $B = \{0, 1\}$ e not: $B \to B : b \to \neg b$.

Notiamo che not è sicuramente iniettiva, e che, poiché è l'unico costruttore, anche la seconda caratteristica delle algebre induttive è rispettata.

Notiamo però che l'algebra non rispetta il terzo requisito. Se consideriamo infatti $\emptyset \subseteq B$, notiamo che not è chiusa rispetto ad esso.

Infatti, l'implicazione $x \in \emptyset \Rightarrow \text{not}(x) \in \emptyset$ risulta vera per falsificazione della premessa (non esistono elementi in $\emptyset$).

$(\emptyset, \text{not})$ è quindi una sotto-algebra induttiva di B, che però è diversa da essa. L'implicazione della terza condizione $(x \in \emptyset \Rightarrow \text{not}(x) \in \emptyset) \Rightarrow \emptyset = B$ è falsa, e (B, not) non è quindi un'algebra induttiva.

1.3. Omomorfismi, lemma di Lambek

digressione - teoria delle categorie

Facciamo una piccola parentesi che introduce alcune nozioni di teoria delle categorie (perché è molto interessante).

La teoria delle categorie studia in modo astratto le strutture matematiche. Una categoria $\mathcal{C}$ consiste di:

- una classe ob($\mathcal{C}$), i cui elementi sono chiamati **oggetti**
- una classe mor($\mathcal{C}$), i cui elementi sono chiamati **morfismi** (o mappe o frecce); ogni morfismo $f : a \to b$ ha associati un unico oggetto sorgente a e un unico oggetto destinazione b.
- per ogni terna di oggetti $a, b, c \in \mathcal{C}$, è definita una funzione $\text{mor}(b, c) \times \text{mor}(a, b) \to \text{mor}(a, c)$ chiamata **composizione di morfismi**. La composizione di $f : b \to c$ con $g : a \to b$ si indica con $f \circ g : a \to c$

  la composizione deve soddisfare i seguenti assiomi:

  (*associatività*): se $f : a \to b,\ g : b \to a$ e $h : c \to d$, allora $h \circ (g \circ f) = (h \circ g) \circ f$

  (*identità*): per ogni oggetto x esiste un morfismo $\text{id}_x : x \to x$ chiamato **morfismo identità**, tale che per ogni morfismo $f : a \to x$ vale $\text{id}_x \circ f = f$ e per ogni morfismo $g : x \to b$ si ha $g \circ \text{id}_x = g$.

Quindi, ogni oggetto è associato ad un unico morfismo identità. Questo permette di dare una definizione di categoria basata esclusivamente sulla classe dei morfismi: gli **oggetti vengono identificati con i corrispondenti morfismi identità**.

All'interno della teoria delle categorie, una funzione iniettiva $f : B \to C$ si chiama **monomorfismo**. Visto che non si possono utilizzare gli elementi per definire l'iniettività, un monomorfismo è descritto come una funzione f tale che:

$$\forall A,\ \forall h, k : A \to B,\quad h \circ f = k \circ f\ \Rightarrow\ h = k$$

(se le funzioni h e k sono identiche ogni volta che vengono composte con f, significa che non ci sono valori in f che sono assunti da più di un elemento di B)

def. 7: Algebre con la stessa segnatura

Due algebre (A, Γ_A) e (B, Γ_B) hanno la stessa segnatura se, sostituendo A con B in tutte le $\gamma_i \in \Gamma_A$, si ottiene Γ_B.

(La segnatura di un'algebra è data dalle segnature delle sue operazioni).

def. 8: Omomorfismo

Date due algebre con la stessa segnatura (A, Γ) e $(B, \Delta = \{\delta_1, \ldots, \delta_k\})$, un omomorfismo è una funzione $f : A \to B$ tale che:

$$\forall i\ \ f(\gamma_i(a_1, \ldots, a_k, k_1, \ldots, k_m)) = \delta_i(f(a_1), \ldots, f(a_k), k_1, \ldots, k_m)$$

(con $k_1, \ldots, k_m$ parametri esterni)

(definizione algebrica: $\forall a, b \in A$, date $\circ$ operazione di A e $\bullet$ operazione di B, si ha $f(a \circ b) = f(a) \bullet f(b)$)

un omomorfismo "rispetta le operazioni"

- nota: la composizione di due omomorfismi è a sua volta un omomorfismo

def. 9: Isomorfismo

Un isomorfismo è un omomorfismo biettivo.

(Due algebre sono isomorfe ($\cong$) quando esiste un isomorfismo tra loro)

theorem 2: Omomorfismo tra algebre con stessa segnatura

Sia A un'algebra induttiva. Per ogni algebra B (non necessariamente induttiva) con la stessa segnatura, esiste un **unico omoformismo** $A \to B$.

theorem 3: Lemma di Lambek

Due algebre induttive A e B con la **stessa segnatura** sono necessariamente **isomorfe**.

proof!

- Siccome A è un'algebra induttiva, $\exists!$ omomorfismo $f : A \to B$.
- Allo stesso modo, $\exists!$ omomorfismo $g : B \to A$.
- Componendo i due omomorfismi, si ottiene un omomorfismo $g \circ f$ con segnatura $A \to A$.
- Sappiamo che per ogni algebra esiste l'omomorfismo "identità".
- Sappiamo anche, per il teorema sopra, che esiste un unico omomorfismo $A \to A$.

  Ne segue necessariamente che $g \circ a = \mathrm{Id}_A$. (lo stesso discorso si applica a $f \circ g = \mathrm{Id}_B$)
- $g \circ f = \mathrm{Id} \iff g = f^{-1}$, quindi g e f sono funzioni invertibili (= biettive) $\Rightarrow$ g, f sono isomorfismi $\Rightarrow$ $A \cong B$

2. Espressioni, linguaggi

Definiamo un **linguaggio** L come insieme di stringhe.

Per descrivere la sintassi di linguaggi formali (la grammatica), usiamo la BNF (Backus-Naur Form), con questa sintassi:

```
<simbolo> ::= __espressione__
```

Esempio: prendiamo come esempio questa grammatica:

$$M, N ::= 5 \mid 7 \mid M + N \mid M * N$$

Le espressioni che seguono questa grammatica, sono del tipo:

- ”5” o ”7”
- un’espressione $M + N$ o $M * N$, in cui M e N rispettano a loro volta la grammatica

Introduciamo una funzione $eval : L \to \mathbb{N}$, che valuta le espressioni del linguaggio:

- $eval(5) = 5$
- $eval(7) = 7$
- $eval(M + N) = eval(M) + eval(N)$
- $eval(M * N) = eval(M) * eval(N)$

Possiamo notare subito che $(L, eval)$ non è un’algebra induttiva. Infatti, una stringa come ”$5 + 7 * 5$” potrebbe essere stata generata in due modi diversi: $(5 + 7) * 5$ e $5 + (7 * 5)$.

Possiamo però stipulare che sia induttiva. Ci basta infatti considerare $+$, $*$, 5 e 7 come costruttori dell’algebra. In questo modo, $(5 + 7) * 5$ risulta essere un oggetto diverso da $5 + (7 * 5)$. È quindi possibile dimostrare che $(L, 5, 7, +, *)$ è un’algebra induttiva.

2.1. Exp

def. 10: Linguaggio Exp

Introduciamo il linguaggio Exp, con grammatica:

$$M, N = k \mid x \mid M + N \mid let\ x = M\ in\ N$$

dove:

- $k \in Val = \{0, 1, \dots\}$ è una costante
- $x \in Var$ è una variabile
- $M + N : Exp \times Exp \to Exp$ è la somma tra due espressioni
- $let : Var \times Exp \times Exp \to Exp$ assegna alla variabile x il valore M all’interno di N

esempi:

- $let\ x = 3\ in\ x + x + 2$ viene valutata come 8
- $let\ x = 3\ in\ 12$ viene valutata come 12

Questo linguaggio causa però facilmente ambiguità. Per esempio, come valutiamo un'espressione come $let\ x = 3\ in\ let\ y = x\ in\ let\ x = 5\ in\ y$?

Per esplicitare la struttura del termine, è necessario legare le occorrenze delle variabili alle dichiarazioni.

def. 11: Variabili libere, legate, scope

Si dice che un'occorrenza di una variabile x è **libera** in un termine t quando non compare nel corpo di N nessun sottotermine di t nella forma $let\ x = M\ in\ N$ (quindi, quando non le viene assegnato un valore).

Ogni occorrenza libera di x in un termine N si dice **legata** (bound) alla dichiarazione di x nel termine $let\ x = M\ in\ N$.

Lo scope di una dichiarazione è l'insieme delle occorrenze libere di x in N.

Lo **scope di una variabile** è la porzione di programma all'interno della quale una variabile può essere riferita.

Introduciamo una funzione $free : Exp \to \mathcal{P}(Var)$, che restituisce l'insieme delle variabili libere di un'espressione:

$$\begin{aligned} free(k) &= \emptyset \\ free(x) &= \{x\} \\ free(M + N) &= free(M) \cup free(N) \\ free(let\ x = M\ in\ N) &= free(M) \cup (free(N) - \{x\}) \end{aligned}$$

(eliminiamo la x, dalle variabili libere in N perché viene dichiarata dal $let\ x$, ma non la eliminiamo da M perché potrebbe comparire al suo interno come variabile libera, e M non fa parte dello scope di $let\ x$ (esempio: in $let\ x = x\ in\ x$, la x è libera perché compare libera in $= x$))

esempio: $free(let\ x = 7\ in\ x + y) = \{y\}$

2.1.1. Semantica operazionale

Vogliamo introdurre nel linguaggio Exp il concetto di "quanto fa?" (valutazione di un'espressione).

Per farlo, abbiamo bisogno di definire un ambiente all'interno del quale valutare le espressioni (stile operazionale, "structural operational semantics").

def. 12: Ambienti

Un **ambiente** è una funzione parziale (funzione non necessariamente definita su tutti gli elementi del dominio) con dominio finito che associa dei valori ad un insieme finito di variabili.

$$E : Var \xrightarrow{fin} Val$$

Scriviamo gli ambienti come insiemi di coppie. Per esempio, l'ambiente E in cui z vale 3 e y vale 9 è indicato con $\{(z, 3), (y, 9)\}$.

Notiamo che, essendo E una funzione parziale, il dominio $dom(E)$ è un sottoinsieme finito di Var.

def. 13: Insieme di ambienti

Env è definito come l'insieme degli ambienti di Exp.

Gli ambienti si possono **concatenare** in questo modo:

$$(E_1E_2)(x) = \begin{cases} E_2(x) & \text{se } x \in dom(E_2) \\ E_1(x) & \text{altrimenti} \end{cases}$$

Per esempio, $\{(z,3),(y,9)\}\{(z,4)\}(z) = 4$ e $\{(z,3),(y,9)\}\{(z,4)\}(x)$ è indefinito.

def. 14: Semantica operazionale di Exp

La **semantica operazionale** di Exp è una relazione

$$\rightsquigarrow \subseteq Exp \times Env \times Val$$

in cui $(M, E, v) \in \rightsquigarrow \iff$ il programma M, nell'ambiente E, produce il valore v.

Un'asserzione di appartenenza $(M, E, v) \in \rightsquigarrow$ viene chiamata *giudizio operazionale*, e si scrive

$$E \vdash M \rightsquigarrow v$$

Questa relazione è definita dalle seguenti **regole**:

$$E \vdash k \rightsquigarrow k \quad [const]$$

(in ogni ambiente E, una costante k vale k)

$$E \vdash x \rightsquigarrow v \quad \text{se } v = E(x) \quad [var]$$

(una variabile x vale v se la funzione ambiente $E(x)$ le associa il valore v)

$$\frac{E \vdash M \rightsquigarrow v \quad E \vdash N \rightsquigarrow w}{E \vdash M + N \rightsquigarrow v + w} \quad [plus]$$

(se nello stesso ambiente M vale v e N vale w, $M + N$ varrà $u + w$)

$$\frac{E \vdash M \rightsquigarrow v_1 \quad E\{(x, v_1)\} \vdash N \rightsquigarrow v_2}{E \vdash \ let\ x = M\ in\ N \rightsquigarrow v_2} \quad [let]$$

(essenzialmente, per valutare una let, si:

- valuta M $\quad (E \vdash M \rightsquigarrow v_1)$
- si "associa" il risultato v_1 a x, concatenando (x, v_1) all'ambiente
- e si valuta N nel nuovo ambiente)

Notiamo che si utilizza la relazione $\rightsquigarrow$ e non una una funzione $Exp \times Env \to Val$, perché si potrebbe avere più di un risultato (per esempio nel caso del multithreading, in cui un diverso ordine di esecuzione di un programma dà output diversi), o anche nessun risultato (per esempio nel caso in cui in Exp compare una variabile x, che però Env non definisce), entrambi casi non accettati dalla definizione di funzione.

precedenza

Introduciamo un concetto di “precedenza” nella valutazione di un’espressione potenzialmente ambigua; un’espressione del tipo:

$$let\ \ x = 3\ \ in\ \ let\ \ x =\ \ let\ \ y = 2\ \ in\ \ x + y\ \ in\ \ x + 7 + x$$

in assenza di parentesi, va valutata “partendo dall’interno”.

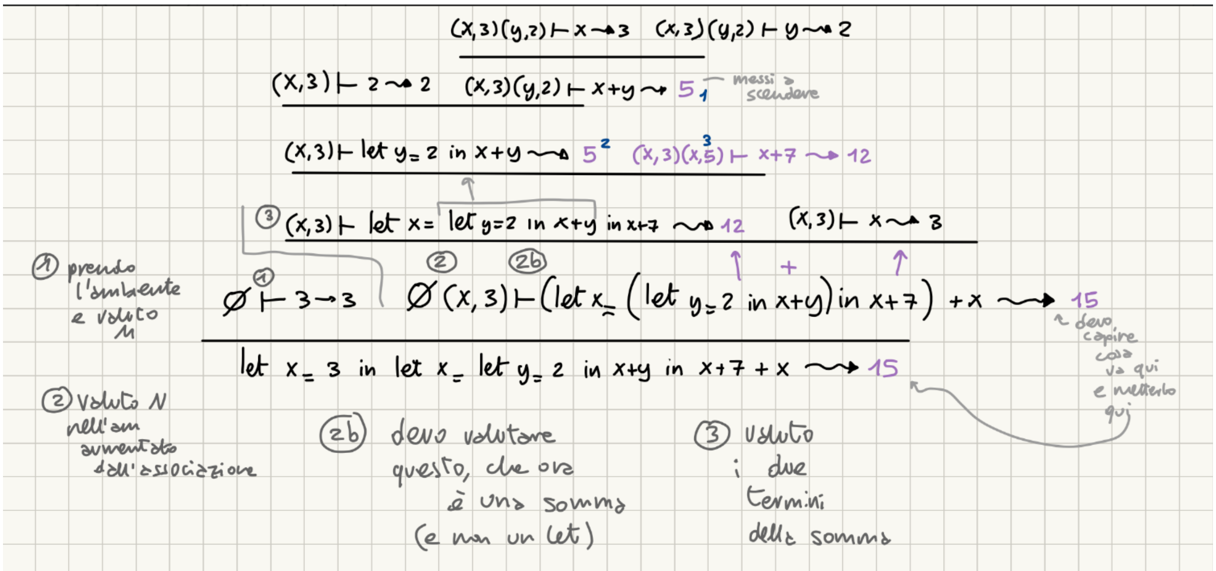
Corrisponde quindi a

$$let\ \ x = 3\ \ in\ \ [\,let\ \ x =\ \ (let\ \ y = 2\ \ in\ \ x + y)\ \ in\ \ x + 7\,] + x$$

E si ha quindi che:

- la x in $x + y$ e quella finale ($+x$) sono quelle valutate dal let iniziale
- il valore della x in $x + 7$ è invece dato dal risultato di $let\ x = (let\ y = 2\ \ in\ \ x + y)$

Facciamo un esempio di valutazione di un’espressione:



(copierò appena ho tempo...)

2.2. Valutazioni Eager e Lazy

La valutazione utilizzata fino a questo momento viene definita **eager**, in quanto valuta N immediatamente (anche nel caso in cui non servisse veramente valutarlo).

Se infatti consideriamo un caso del tipo $let\ =$ [espressione lunghissima] in 7, notiamo immediatamente che la valutazione di N non è necessaria, in quanto l’espressione farà, in ogni caso, 7.

Introduciamo quindi un approccio **lazy**, che consiste nel valutare un termine solo quando (e se) ce n’è veramente bisogno. La valutazione di N in un termine del tipo $let\ x =\ N\ in\ M$ viene rimandata, quindi, al momento in cui ad M (eventualmente) servirà il suo valore.

finire!