



SAPIENZA
UNIVERSITÀ DI ROMA

“SAPIENZA” UNIVERSITÀ DI ROMA
INGEGNERIA DELL'INFORMAZIONE,
INFORMATICA E STATISTICA
DIPARTIMENTO DI INFORMATICA

Linguaggi di Programmazione

Author
Simone Bianco

12 ottobre 2023

Indice

Informazioni e Contatti	1
1 Struttura e Rappresentazione	2
1.1 Algebre induttive	2
1.1.1 Lemma di Lambek	8
1.2 Strutture dati induttive	9
1.2.1 Induzione strutturale	11
1.3 Sintassi astratta	12
2 Paradigma funzionale	14
2.1 <i>Exp</i> : un semplice linguaggio funzionale	14
2.2 Valutazione Eager vs Lazy	18
2.3 Scoping Statico vs Dinamico	20
2.4 <i>Fun</i> : un linguaggio con funzioni	23

Informazioni e Contatti

Appunti e riassunti personali raccolti in ambito del corso di *Linguaggi di Programmazione* offerto dal corso di laurea in Informatica dell'Università degli Studi di Roma "La Sapienza".

Ulteriori informazioni ed appunti possono essere trovati al seguente link:

<https://github.com/Exyss/university-notes>. Chiunque si senta libero di segnalare incorrettezze, migliorie o richieste tramite il sistema di Issues fornito da GitHub stesso o contattando in privato l'autore :

- Email: bianco.simone@outlook.it
- LinkedIn: [Simone Bianco](#)

Gli appunti sono in continuo aggiornamento, pertanto, previa segnalazione, si prega di controllare se le modifiche siano già state apportate nella versione più recente.

Prerequisiti consigliati per lo studio:

Apprendimento del materiale relativo al corso *Algebra*.

Licence:

These documents are distributed under the [GNU Free Documentation License](#), a form of copyleft intended for use on a manual, textbook or other documents. Material licensed under the current version of the license can be used for any purpose, as long as the use meets certain conditions:

- All previous authors of the work must be **attributed**.
- All changes to the work must be **logged**.
- All derivative works must be **licensed under the same license**.
- The full text of the license, unmodified invariant sections as defined by the author if any, and any other added warranty disclaimers (such as a general disclaimer alerting readers that the document may not be accurate for example) and copyright notices from previous versions must be maintained.
- Technical measures such as DRM may not be used to control or obstruct distribution or editing of the document.

1

Struttura e Rappresentazione

1.1 Algebre induttive

Definizione 1: Assiomi di Peano

L'insieme dei numeri naturali \mathbb{N} è definito secondo i seguenti **assiomi di Peano**:

1. $0 \in \mathbb{N}$
2. $n \in \mathbb{N} \implies \text{succ}(n) \in \mathbb{N}$, dove $\text{succ} : \mathbb{N} \rightarrow \mathbb{N}$ è la funzione successore
3. $\forall n, m \in \mathbb{N}, \text{succ}(n) = \text{succ}(m) \implies n = m$, ossia succ è iniettiva
4. $\nexists n \in \mathbb{N} \mid \text{succ}(n) = 0$
5. $\forall S \subseteq \mathbb{N} \mid (0 \in S \wedge (n \in S \implies \text{succ}(n) \in S)) \implies S = \mathbb{N}$

Proposizione 1: Numeri naturali di Von Neumann

I numeri naturali data da Von Neumann, indicati con \mathcal{N} , definiti come:

$$0_{\mathcal{N}} := \{\}$$

$$1_{\mathcal{N}} := \{\{\}\}$$

$$2_{\mathcal{N}} := \{\{\}, \{\{\}\}\}$$

$$3_{\mathcal{N}} := \{\{\}, \{\{\}\}, \{\{\}, \{\{\}\}\}\}$$

...

dove $\text{succ}_{\mathcal{N}} : \mathcal{N} \rightarrow \mathcal{N} : n \mapsto n \cup \{n\}$, soddisfano gli assiomi di Peano

Dimostrazione.

1. $0_{\mathcal{N}} \in \mathcal{N}$ per definizione stessa di \mathcal{N}
2. $n \in \mathcal{N} \implies \text{succ}_{\mathcal{N}}(n) \in \mathcal{N}$ per definizione stessa di $\text{succ}_{\mathcal{N}}$
3. Siano $n, m \in \mathcal{N}$ tali che $n \neq m$. In tal caso, ne segue automaticamente che:

$$n \neq m \implies n \cup \{n\} \neq m \cup \{m\} \iff \text{succ}_{\mathcal{N}}(n) \neq \text{succ}_{\mathcal{N}}(m)$$

Per contro-nominale, dunque, otteniamo che:

$$\text{succ}_{\mathcal{N}}(n) = \text{succ}_{\mathcal{N}}(m) \implies n = m$$

4. Supponiamo per assurdo che $\exists n \in \mathbb{N} \mid \text{succ}_{\mathcal{N}}(n) = 0_{\mathcal{N}}$. In tal caso, avremmo che:

$$\text{succ}(n) = 0_{\mathcal{N}} \iff n \cup \{n\} = 0_{\mathcal{N}} \iff n \cup \{n\} = \{\}$$

ma ciò risulta assurdo poiché implicherebbe che l'insieme $\{\}$ contenga degli elementi. Di conseguenza, l'unica possibilità è che $\nexists n \in \mathbb{N} \mid \text{succ}_{\mathcal{N}}(n) = 0_{\mathcal{N}}$

5. Supponiamo per assurdo che $\exists S \subseteq \mathcal{N} \mid (0_{\mathcal{N}} \in S \wedge (n \in S \implies \text{succ}_{\mathcal{N}}(n) \in S)) \wedge S \neq \mathcal{N}$. Consideriamo quindi $\mathcal{N} - S = \{n_1, \dots, n_k\}$. Per via del secondo assioma, ogni elemento di $\mathcal{N} - S$ deve avere un proprio successore e un proprio predecessore in \mathcal{N} .

Poiché per ipotesi si ha che $n \in S \implies \text{succ}_{\mathcal{N}}(n) \in S$, ne segue che tutti i predecessori degli elementi in $\mathcal{N} - S$ non possano essere in S , poiché altrimenti tali elementi sarebbero in S . Inoltre, poiché $\text{succ}_{\mathcal{N}}$ è iniettiva, ne segue che i successori degli elementi in $\mathcal{N} - S$ non possano essere in S , poiché esiste già un predecessore in S per ogni elemento in S .

Di conseguenza, ogni predecessore ed ogni successore degli elementi di $\mathcal{N} - S$ deve essere in $\mathcal{N} - S$ stesso. Consideriamo quindi (per comodità) la seguente catena di successori in $\mathcal{N} - S$:

$$n_1 \rightarrow n_2 \rightarrow \dots \rightarrow n_k \rightarrow n_1$$

Notiamo a questo punto che:

$$\text{succ}_{\mathcal{N}}^k(n_1) = n_1 \implies n_1 \in n_1$$

contraddicendo gli assiomi insiemistici per cui un insieme non possa essere contenuto in se stesso. Di conseguenza, l'unica possibilità è che $S = \mathcal{N}$

□

Principio 1: Principio di induzione

Sia P una proprietà che vale per $n = 0$. Dato $n \in \mathbb{N}$, se si verifica che la veridicità di P per n implica che P sia vera anche per $n + 1$, allora P vale per tutto \mathbb{N} . In simboli, abbiamo che:

$$\forall P ((P(0) \wedge (P(n) \implies P(n+1)))) \implies \forall m \in \mathbb{N} P(m)$$

Osservazione 1

Il quinto assioma di Peano è equivalente al principio di induzione, poiché basta considerare $S \subseteq \mathbb{N}$ come l'insieme degli elementi per cui vale la proprietà desiderata

Osservazione 2

Dato $k \in \mathbb{N}$, il principio di induzione può essere utilizzato per dimostrare che una proprietà P valga $\forall n \in \mathbb{N} \mid n \geq k$. In altre parole, non è necessario che il principio valga per tutti i naturali a partire da 0.

Dimostrazione.

- Definendo una proprietà Q tale che $P(n) = Q(n - k)$, si ha che:

$$\forall n - k \in \mathbb{N} \quad Q(n - k) \iff P(n)$$

dunque applicare il principio di induzione per P partendo da k equivale ad applicare il principio di induzione per Q partendo da 0, rispettando quindi il quinto assioma di Peano

□

Definizione 2: Insieme unità

Definiamo come **insieme unità** l'insieme $\mathbb{1} = \{()\}$, ossia l'insieme composto da una zerupla

Definizione 3: Funzione nullaria

Definiamo una funzione $f : \mathbb{1} \rightarrow S$, dunque avente $\mathbb{1}$ come dominio, come **funzione nullaria** (o funzione costante).

Inoltre, per comodità, indichiamo $f(x)$ direttamente con f , poiché $x = ()$

Esempio:

- Data la funzione zero : $\mathbb{1} \rightarrow \mathbb{N} : x \mapsto 0$, indichiamo zero(x) direttamente come zero

Osservazione 3

Una funzione nullaria è sempre **iniettiva** in quanto esiste un solo elemento nel dominio.

Definizione 4: Segnatura di una funzione

Data una funzione f definiamo $f : D \rightarrow C$ come **segnatura di f** dove D è il **dominio di f** e C è il **codominio di f**

Definizione 5: Algebra

Definiamo come **algebra** (o struttura algebrica) una n -upla $(A, \gamma_1, \dots, \gamma_n)$ dove A è un insieme non vuoto, detto **dominio**, e $\gamma_1, \dots, \gamma_n$ sono delle operazioni definite su A stesso.

Esempi:

- La coppia $(\mathbb{N}, \text{succ})$ è un'algebra
- La coppia $(\mathbb{N}, \text{zero})$ è un'algebra

Definizione 6: Segnatura di un'algebra

Data un'algebra $(A, \gamma_1, \dots, \gamma_n)$, definiamo come **segnatura dell'algebra** l'insieme delle segnature delle operazioni definite su essa

Definizione 7: Segnature equivalenti

Date due algebre $(A, \gamma_1, \dots, \gamma_n)$ e $(B, \delta_1, \dots, \delta_n)$, definiamo le segnature di tali algebre come **equivalenti** se per ogni operazione γ definita su A esiste un'operazione δ definita su B per cui invertendo B con A all'interno della segnatura di δ si ottiene la segnatura di γ

Esempio:

- Date le due algebre $(\mathbb{N}, \text{zero}, \text{succ})$ e $(\mathcal{N}, \text{zero}_{\mathcal{N}}, \text{succ}_{\mathcal{N}})$, le segnature di tali algebre sono equivalenti poiché:
 - La segnatura di $\text{zero} : \mathbb{1} \rightarrow \mathbb{N}$ è equivalente alla segnatura di $\text{zero}_{\mathcal{N}} : \mathbb{1} \rightarrow \mathcal{N}$
 - La segnatura di $\text{succ} : \mathbb{N} \rightarrow \mathbb{N}$ è equivalente alla segnatura di $\text{succ}_{\mathcal{N}} : \mathcal{N} \rightarrow \mathcal{N}$

Definizione 8: Algebra induttiva e Costruttori

Definiamo l'algebra $(A, \gamma_1, \dots, \gamma_n)$ come **induttiva** (o **iniziale**) se:

- $\gamma_1, \dots, \gamma_n$ sono iniettive
- $\forall i \neq j \quad \text{im}(\gamma_i) \cap \text{im}(\gamma_j) = \emptyset$, ossia le immagini delle operazioni sono due a due disgiunte
- $\forall S \subseteq A \quad (\forall i \in [1, n], a_1, \dots, a_k \in S \quad \gamma_i(a_1, \dots, a_k) \in S) \implies S = A$, ossia è soddisfatto il principio di induzione per ogni operazione

Inoltre, definiamo $\gamma_1, \dots, \gamma_n$ come **costruttori di A** .

Esempi:

- L'algebra $(\mathbb{N}, +)$ non è un'algebra induttiva poiché $+: \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ non è iniettiva
- L'algebra $(\mathbb{N}, \text{succ}, \text{zero})$ è un'algebra induttiva poiché:
 - succ risulta essere iniettiva grazie al secondo assioma di Peano, mentre zero risulta essere iniettiva poiché funzione nullaria
 - $\text{im}(\text{succ}) \cap \text{im}(\text{zero}) = (\mathbb{N} - \{0\}) \cap \{0\} = \emptyset$
 - Sia $S \subseteq \mathbb{N}$ tale che $\forall x \in S \quad \text{succ}(x) \in S$ e $\text{zero} \in S$. Preso $x \in \mathbb{N}$, possiamo esprimere x come $x = \text{succ}(\text{succ}(\dots(\text{zero})))$.

Di conseguenza, poiché S è chiuso per succ e zero, otteniamo che:

- * $\text{zero} \in S \implies \text{succ}(\text{zero}) \in S$
- * $\text{succ}(\text{zero}) \in S \implies \text{succ}(\text{succ}(\text{zero})) \in S$
- * ...
- * $\text{succ}(\dots(\text{zero})) \in S \implies x = \text{succ}(\text{succ}(\dots(\text{zero}))) \in S$

Di conseguenza, otteniamo che $A \subseteq S$ e dunque che $S = A$

Osservazione 4

Equivalentemente, la terza condizione necessaria delle algebre induttive può essere considerata come

$$\nexists S \subsetneq A \mid (S, \gamma_1, \dots, \gamma_n) \text{ è algebra induttiva}$$

Definizione 9: Omomorfismo

Date due strutture algebriche $(A, \gamma_1, \dots, \gamma_k)$ e $(B, \delta_1, \dots, \delta_k)$ dello stesso tipo, definiamo $f: A \rightarrow B$ come **omomorfismo** se

$$\forall a_1, \dots, a_n \in A, i \in [1, k] \quad f(\gamma_i(a_1, \dots, a_k)) = \delta_i(f(a_1), \dots, f(a_k))$$

Esempi:

- Date le due algebre $(\mathbb{N}, \text{succ}, +)$ e $(\mathcal{N}, \text{succ}_{\mathcal{N}}, +_{\mathcal{N}})$, affinché la funzione $f : \mathbb{N} \rightarrow \mathcal{N}$ sia un omomorfismo è necessario che:

$$f(\text{succ}(n)) = \text{succ}_{\mathcal{N}}(f(n)) \quad f(n + m) = f(n) +_{\mathcal{N}} f(m)$$

- Date le due algebre $(\mathbb{R}, +)$ e $(\mathbb{R}_{>0}, \cdot)$, la funzione $\exp : \mathbb{R} \rightarrow \mathbb{R}_{>0} : x \mapsto e^x$ è un omomorfismo:

$$\exp(x + y) = e^{x+y} = e^x e^y = \exp(x)\exp(y)$$

Definizione 10: Isomorfismo

Definiamo come **isomorfismo** un omomorfismo biiettivo. Inoltre, definiamo due algebre $(A, \gamma_1, \dots, \gamma_n)$, $(B, \delta_1, \dots, \delta_n)$ come **isomorfe**, indicato con $A \cong B$, se esiste un isomorfismo tra loro.

Osservazione 5

Data una funzione $f : A \rightarrow B$, si ha che:

$$f \text{ è biettiva} \iff \exists f^{-1} : B \rightarrow A$$

(*dimostrazione omessa*)

Osservazione 6

Data una funzione $f : A \rightarrow B$, si ha che:

$$f \text{ è un isomorfismo} \iff f^{-1} \text{ è un isomorfismo}$$

(*dimostrazione omessa*)

Esempio:

- Date le due algebre $(\mathbb{R}, +)$ e $(\mathbb{R}_{>0}, \cdot)$, la funzione $\exp : \mathbb{R} \rightarrow \mathbb{R}_{>0} : x \mapsto e^x$ è un isomorfismo, poiché:
 - \exp è un omomorfismo
 - $\exists \ln : \mathbb{R}_{>0} \rightarrow \mathbb{R} \mid \ln(\exp(x)) = x$, dunque f è biettiva.

1.1.1 Lemma di Lambek

Lemma 1

Data un'algebra induttiva $(A, \gamma_1, \dots, \gamma_n)$, per ogni algebra $(B, \delta_1, \dots, \delta_n)$ con la stessa segnatura di A si ha che

$$\exists! \text{ omomorfismo } f : A \rightarrow B$$

Nota: l'algebra di B non deve necessariamente essere induttiva

(*dimostrazione omessa*)

Teorema 1: Lemma di Lambek (versione ridotta)

Date due algebre induttive $(A, \gamma_1, \dots, \gamma_n)$ e $(B, \delta_1, \dots, \delta_n)$ si ha che $A \cong B$

Dimostrazione.

- Per il lemma precedente, si ha che:

$$\exists! \text{ omomorfismo } f : A \rightarrow B$$

$$\exists! \text{ omomorfismo } g : B \rightarrow A$$

- Consideriamo quindi la funzione $g \circ f : A \rightarrow A$ ($g(f(x))$) e verifichiamo che essa sia un omomorfismo

$$g \circ f(x + y) = g(f(x + y)) = g(f(x) + f(y)) = g(f(x)) + g(f(y)) = g \circ f(x) + g \circ f(y)$$

- Tuttavia, per ogni algebra esiste sempre l'isomorfismo identità $\text{id} : A \rightarrow A : x \mapsto x$ e poiché per il lemma precedente esiste necessariamente un unico omomorfismo tra A e A , ne segue necessariamente che $g \circ f = \text{id}$
- Di conseguenza, si ha che

$$g \circ f = \text{id} \iff g = f^{-1} \implies g, f \text{ biettive} \implies g, f \text{ isomorfismi} \implies A \cong B$$

□

Esempio:

- Date le due algebre induttive $(\mathbb{N}, \text{zero}, \text{succ})$ e $(\mathcal{N}, \text{zero}_{\mathcal{N}}, \text{succ}_{\mathcal{N}})$ sono isomorfe tra loro poiché aventi la stessa segnatura algebrica
- Difatti, come già dimostrato, \mathbb{N} e \mathcal{N} sono solamente due modi diversi per rappresentare lo stesso identico concetto algebrico

1.2 Strutture dati induttive

Definizione 11: Insieme delle liste finite

Definiamo $\text{List}\langle T \rangle$ come l'insieme delle liste finite di elementi di T

Esempio:

- Dato $\text{List}\langle \text{Int} \rangle$, si ha che $[3 \rightarrow 5 \rightarrow 1] \in \text{List}\langle \text{Int} \rangle$

Proposizione 2: Algebra induttiva delle liste finite

La tripla $(\text{List}\langle T \rangle, \text{empty}, \text{cons})$, dove:

- $\text{empty} : \mathbb{1} \rightarrow \text{List}\langle T \rangle : x \mapsto []$ è la funzione nullaria che restituisce la **lista vuota**
- $\text{cons} : \text{List}\langle T \rangle \times T \rightarrow \text{List}\langle T \rangle : x, ([x_1 \rightarrow \dots \rightarrow x_n]) \mapsto [x \rightarrow x_1 \rightarrow \dots \rightarrow x_n]$ è la funzione di **costruzione delle liste**

è un'algebra induttiva

Dimostrazione.

1. La funzione empty risulta essere iniettiva poiché nullaria.

Dati $\ell_1, \ell_2 \in \text{List}\langle T \rangle$ e $x_1, x_2 \in T$, supponiamo che:

$$\text{cons}(y_1, \ell_1) = \text{cons}(y_2, \ell_2) = [x_1 \rightarrow x_2 \rightarrow \dots \rightarrow x_n]$$

Per definizione stessa di cons , si ha che:

$$\text{cons}(y_1, \ell_1) = \text{cons}(y_2, \ell_2) = [x \rightarrow x_1 \rightarrow \dots \rightarrow x_n]$$

$$\implies y_1 = y_2 = x, \ell_1 = \ell_2 = [x_1 \rightarrow \dots \rightarrow x_n]$$

dunque anche cons risulta iniettiva

2. $\text{im}(\text{empty}) \cap \text{im}(\text{cons}) = \{[]\} \cap (\text{List}\langle T \rangle - \{[]\}) = \emptyset$
3. Sia $S \subseteq \text{List}\langle T \rangle$ tale che $\forall x \in T, \ell \in \text{List}\langle T \rangle \text{ } \text{cons}(x, \ell) \in S$ e $\text{empty} \in S$.

Preso $\ell := [x_1 \rightarrow x_2 \rightarrow \dots \rightarrow x_n] \in \text{List}\langle T \rangle$, possiamo esprimere ℓ come

$$\ell = \text{cons}(x_1, \text{cons}(x_2, \dots \text{cons}(x_n, \text{empty})))$$

Di conseguenza, poiché S è chiuso per cons e empty e poiché $\text{empty} \in S$, otteniamo che ogni valore della catena sia contenuto in S , implicando che $x \in S$ e quindi che $\text{List}\langle T \rangle \subseteq S$, concludendo che $S = \text{List}\langle T \rangle$

□

Osservazione 7

La tripla $(\text{List}\langle T \rangle_\infty, \text{empty}, \text{cons})$, dove $\text{List}\langle T \rangle_\infty$ è l'insieme delle liste infinite di elementi di T **non è un'algebra induttiva**, poiché $\text{List}\langle T \rangle \subsetneq \text{List}\langle T \rangle_\infty$ e poiché $(\text{List}\langle T \rangle, \text{empty}, \text{cons})$ è un'algebra induttiva

Osservazione 8

Tramite i costruttori di un'algebra induttiva è possibile definire le ulteriori operazioni "aggiuntive" di tale algebra

Esempio:

- Data l'algebra induttiva $(\text{List}\langle T \rangle, \text{empty}, \text{cons})$, definiamo la seguente operazione

$$\text{concat} : \text{List}\langle T \rangle \times \text{List}\langle T \rangle \rightarrow \text{List}\langle T \rangle$$

dove:

$$\begin{cases} \text{concat}(\text{empty}, \ell) = \ell \\ \text{concat}(\text{cons}(n, \ell), \ell') = \text{cons}(n, \text{concat}(\ell, \ell')) \end{cases}$$

- Ad esempio, in $\text{List}\langle \text{Int} \rangle$, abbiamo che:

$$\begin{aligned} \text{concat}([1 \rightarrow 5], [7 \rightarrow 2]) &= \text{concat}(\text{cons}(1, [5]), [7 \rightarrow 2]) = \text{cons}(1, \text{concat}([5], [7 \rightarrow 2])) = \\ &= \text{cons}(1, \text{concat}(\text{cons}(5, \text{empty}), [7 \rightarrow 2])) = \text{cons}(1, \text{cons}(5, \text{concat}(\text{empty}, [7 \rightarrow 2]))) = \\ &= \text{cons}(1, \text{cons}(5, [7 \rightarrow 2])) = \text{cons}(1, [5 \rightarrow 7 \rightarrow 2]) = [1 \rightarrow 5 \rightarrow 7 \rightarrow 2] \end{aligned}$$

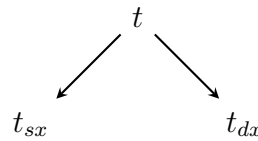
Definizione 12: Insieme degli alberi binari finiti

Definiamo **BinTree** come l'insieme degli alberi binari finiti

Proposizione 3: Algebra induttiva degli alberi binari finiti

La tripla $(\text{BinTree}, \text{leaf}, \text{branch})$, dove:

- $\text{leaf} : \mathbb{1} \rightarrow \text{BinTree} : x \mapsto \circ$ è la funzione nullaria che restituisce una **foglia**
- $\text{branch} : \text{BinTree} \times \text{BinTree} \rightarrow \text{BinTree} : (t_{sx}, t_{dx}) \mapsto t$ è la funzione di **costruzione dei rami**, ossia tale che



è un'algebra induttiva

(*dimostrazione omessa*)

Esempio:

- Il seguente albero



corrisponde a:

$$a = \text{branch}(\text{leaf}, \text{branch}(\text{leaf}, \text{leaf}))$$

1.2.1 Induzione strutturale**Definizione 13: Induzione strutturale**

Definiamo come **induzione strutturale** il metodo dimostrativo generalizzante il principio di induzione e basato sulle proprietà di un'algebra induttiva.

In particolare, viene ipotizzato che una proprietà P valga per ogni argomento di ogni costruttore dell'algebra e tramite il terzo assioma viene dimostrato che tale proprietà valga per tutti gli elementi dell'algebra stessa

Teorema 2: Relazione tra nodi e foglie

Dato $t \in \text{BinTree}$ avente n foglie, il numero di nodi di t è pari a $2n - 1$

Dimostrazione per induzione strutturale.

- Definiamo l'operazione

$$\text{leaves} : \text{BinTree} \rightarrow \mathbb{N} : t \mapsto \text{Numero di foglie in } t$$

dove:

$$\begin{cases} \text{leaves}(\text{leaf}) = 1 \\ \text{leaves}(\text{branch}(b_1, b_2)) = \text{leaves}(b_1) + \text{leaves}(b_2) \end{cases}$$

- Dato $t \in \text{BinTree}$, sia k il numero di nodi di t e sia $n = \text{leaves}(t)$

Caso base. Se $t = \text{leaf}$, allora t è composto da $k = 1$ nodi e $n = \text{leaves}(\text{leaf}) = 1$ foglie. Difatti, si ha che $k = 1 = 2n - 1$

Ipotesi induttiva. Ogni argomento t' di ogni costruttore possiede $k' = 2\text{leaves}(t') - 1$ nodi

Passo induttivo. Se $t \neq \text{leaf}$, allora $\exists t_1, t_2 \in \text{BinTree} \mid t = \text{branch}(t_1, t_2)$ dove t_1 e t_2 possiedono rispettivamente k_1 e k_2 nodi. Inoltre, si ha che $k = k_1 + k_2 + 1$

In quanto t_1 e t_2 sono argomenti del costruttore `branch`, per ipotesi induttiva si ha che:

$$\begin{aligned} k &= k_1 + k_2 + 1 = 2\text{leaves}(t_1) - 1 + 2\text{leaves}(t_2) - 1 + 1 = 2(\text{leaves}(t_1) + \text{leaves}(t_2)) - 1 = \\ &= 2(\text{leaves}(\text{branch}(t_1, t_2))) - 1 = 2(\text{leaves}(t)) - 1 \end{aligned}$$

□

1.3 Sintassi astratta

Definizione 14: Linguaggio

Definiamo come **linguaggio** un insieme di stringhe

Definizione 15: Grammatica

Definiamo come **grammatica** un insieme di regole, dette **termini**, che definiscono come poter manipolare le stringhe di un linguaggio.

La **forma di Backus-Naur** è una notazione utilizzata per descrivere grammatiche ed è definita come:

$$\langle \text{symbol} \rangle ::= _ \text{expression} _$$

dove:

- $\langle \text{symbol} \rangle$ è una simbolo non-terminale espresso dalla grammatica
- L'operatore $::=$ indica che ciò che si trova alla sua sinistra possa essere sostituito con ciò che si trova alla sua destra
- $\langle _ \text{expression} _ \rangle$ consiste in una o più sequenze di simboli terminali o non-terminali dove ogni sequenza è separata da una barra verticale (ossia $|$) indicante una scelta possibile per l'operatore $::=$

Esempio:

- Consideriamo il linguaggio L espresso dalla grammatica:

$$M, N ::= 0 \mid 1 \mid \dots \mid M + N \mid M * N$$

Tale grammatica indica che i simboli non-terminali M e N possono essere sostituiti con:

- Un numero naturale
- Un'espressione $M + N$ o $M * N$ dove M e N sono due ulteriori simboli terminali o non-terminali

- Ad esempio, abbiamo che la stringa "5 + 7" sia ben definita dalla grammatica, mentre la stringa "5 + +" non lo sia

Definizione 16: Sintassi astratta

La **sintassi astratta** di un linguaggio è una definizione induttiva di un insieme T di termini, permettendo di definire strutture algebriche senza dover necessariamente definire concretamente le sue operazioni

Esempio:

- Consideriamo ancora il linguaggio L definito dalla grammatica

$$M, N ::= 0 \mid 1 \mid \dots \mid M + N \mid M * N$$

- Definiamo quindi la funzione $\text{eval} : L \rightarrow \mathbb{N}$ in grado di valutare le espressioni del linguaggio:

$$\text{eval}("0") = 0$$

$$\text{eval}("1") = 1$$

...

$$\text{eval}("M + N") = \text{eval}("M") + \text{eval}("N")$$

$$\text{eval}("M * N") = \text{eval}("M") * \text{eval}("N")$$

- Notiamo quindi che la grammatica definisca in modo astratto (ma concretamente tramite eval) le seguenti operazioni:

$$0 : \mathbb{1} \rightarrow \mathbb{N} : x \mapsto 0$$

$$1 : \mathbb{1} \rightarrow \mathbb{N} : x \mapsto 1$$

...

$$\text{plus} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} : (m, n) \mapsto m + n$$

$$\text{times} : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} : (m, n) \mapsto m \cdot n$$

- Notiamo però che le operazioni plus e times non risultano essere né iniettive né con immagini disgiunte. Di conseguenza, la funzione eval non ci permette di definire un'algebra induttiva.
- Tuttavia, per tale linguaggio è comunque possibile definire (in qualche modo, ad esempio fissando una precedenza per le operazioni rompendo proprietà come l'associatività e la commutatività) una funzione che possa descrivere un'algebra induttiva.

Teorema 3: Algebra induttiva dei termini

Dato un linguaggio L con una sintassi astratta con termini definiti in T , esiste sempre un'algebra induttiva (T, α) . Di conseguenza, **tutte le proprietà** di un linguaggio sono dimostrabili tramite l'induzione strutturale sulla sua algebra dei termini.

(*dimostrazione omessa*)

2

Paradigma funzionale

2.1 *Exp*: un semplice linguaggio funzionale

Definizione 17: Il linguaggio *Exp*

Definiamo come *Exp* il linguaggio rappresentato dalla seguente grammatica:

$$M, N ::= k \mid x \mid M + N \mid \text{let } x = M \text{ in } N$$

dove:

- $k \in \{0, 1, \dots\}$ ossia è una **costante**
- $x \in Var = \{x, y, z, \dots\}$ ossia è una **variabile**
- $+$: $Exp \times Exp \rightarrow Exp$ la quale **somma le due espressioni**
- $\text{let} : Var \times Exp \times Exp \rightarrow Exp$ la quale **assegna** alla variabile x l'espressione M all'interno della **valutazione** di N . Inoltre, x prende il nome di variabile locale all'interno di N .
- $Val = \{0, 1, \dots\}$ è l'**insieme dei valori** in cui un'espressione può essere valutata

Esempi:

- L'espressione $\text{let } x = 3 \text{ in } x + 1$ indica che la variabile x assuma valore 3 all'interno della valutazione di $x + 1$. Di conseguenza, il risultato della valutazione dell'espressione è 4
- L'espressione $\text{let } x = 3 \text{ in } 7$ viene valutata come 7
- L'espressione $\text{let } y = 9 \text{ in } (\text{let } x = (\text{let } y = 2 \text{ in } y + 1) \text{ in } x + y)$ viene valutata come 12 (si consiglia di cercare di capire come le clausole interne sovrascrivano i valori delle clausole esterne. Se ciò risultasse complesso, più avanti verranno forniti strumenti matematici per valutare in modo corretto le clausole *let* annidate)

Definizione 18: Scope di una variabile

Data un'espressione e una variabile x , definiamo come **scope di x** la porzione la porzione dell'espressione all'interno della quale una variabile può essere riferita, ossia per cui ne è definito il valore.

Una variabile il cui valore non è assegnato in una porzione dell'espressione viene detta **variabile libera**

Definizione 19: Variabile libera

Data un'espressione $expr \in Exp$, definiamo $x \in expr$ come **libera** se x non ha un valore assegnato durante la valutazione di $expr$.

Esempio:

- L'espressione $let\ x = (let\ y = 2\ in\ y + 1)\ in\ x + y$ non è coerente con la grammatica di *Exp*, poiché y non è definito durante la valutazione di $x + y$. Di conseguenza, non è possibile valutare tale espressione.

Proposizione 4: Calcolo delle variabili libere

Dato il linguaggio *Exp*, la funzione

$$free : Exp \rightarrow \mathcal{P}(Var)$$

restituisce l'insieme di tutte le **variabili libere** di un'espressione dove:

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

Nota: $\mathcal{P}(Var)$ è l'insieme delle parti di Var , ossia l'insieme contenente tutti i suoi sottoinsiemi possibili

Esempio:

- Riprendendo l'esempio precedente, notiamo che:

$$\begin{aligned} & free(let\ x = (let\ y = 2\ in\ y + 1)\ in\ x + y) = \\ & = free(let\ y = 2\ in\ y + 1) \cup (free(x + y) - \{x\}) = \\ & = free(let\ y = 2\ in\ y + 1) \cup ((free(x) \cup free(y)) - \{x\}) = \\ & = free(let\ y = 2\ in\ y + 1) \cup ((\{x\} \cup \{y\}) - \{x\}) = \\ & = free(let\ y = 2\ in\ y + 1) \cup \{y\} = \end{aligned}$$

$$\begin{aligned}
 &= (\text{free}(2) \cup (\text{free}(y+1) - \{y\})) \cup \{y\} = \\
 &= ((\text{free}(y)) - \{y\}) \cup \{y\} = \\
 &= \{y\}
 \end{aligned}$$

dunque l'espressione è invalutabile

Definizione 20: Insieme degli ambienti

Dato il linguaggio *Exp*, definiamo come **insieme degli ambienti di *Exp***, indicato con *Env*, l'insieme delle funzioni parziali (ossia non necessariamente definite su tutto il dominio) che associano ogni variabile al proprio valore:

$$Env = \{f \mid f : Var \xrightarrow{fin} Val\}$$

Definizione 21: Concatenazione di ambienti

Dato il linguaggio *Exp*, definiamo l'operazione di **concatenazione di ambienti**, ossia:

$$\cdot : Env \times Env \rightarrow Env$$

dove:

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

Nota: tale operazione può essere interpretata come una sovrascrittura in E_1 di tutte le variabili definite in E_2

Esempio:

- Dati gli ambienti $E_1 = \{(x, 4), (y, 3)\}$ e $E_2 = \{(x, 5)\}$, si ha che

$$(E_1 E_2)(x) = 5$$

$$(E_1 E_2)(y) = 3$$

Proposizione 5: Regola di inferenza

Data la proposizione:

$$\text{Premessa 1} \wedge \dots \wedge \text{Premessa n} \implies \text{Conclusione}$$

definiamo come **regola di inferenza** la notazione alternativa:

$$\frac{\text{Premessa 1} \quad \dots \quad \text{Premessa n}}{\text{Conclusione}}$$

Definizione 22: Semantica operativa

Data la seguente relazione detta **semantica operativa**, ossia:

$$\leadsto \subseteq Env \times Exp \times Val$$

definiamo come **giudizio operativo** la tripla $(E, M, v) \in \leadsto$ descritta dalla notazione

$$E \vdash M \leadsto v$$

la quale viene letta come "nell'ambiente E , M viene valutato come v ".

Proposizione 6: Regole operative di *Exp*

Definiamo come **regole operative** le regole di inferenza che dettano le valutazioni effettuate dalla semantica operativa:

- Per le **costanti** si ha che:

$$\forall E \in Env \quad E \vdash k \leadsto k$$

- Dato $E \in Env$, per le **variabili** si ha che:

$$x \in dom(E) \wedge E(x) = v \implies E \vdash x \leadsto v$$

- Dato $E \in Env$, per la **somma** si ha che:

$$u = v + v' \implies \frac{E \vdash M \leadsto v \quad E \vdash N \leadsto v'}{E \vdash M + N \leadsto u}$$

- Per l'espressione **let** si ha che:

$$\frac{E \vdash M \leadsto v \quad E\{(x, v)\} \vdash N \leadsto v'}{E \vdash let \ x = M \ in \ N \leadsto v'}$$

Osservazione 9: Ambiente iniziale

A meno che non vi siano variabili esternamente assegnate, all'interno di un'espressione l'**ambiente iniziale** corrisponde sempre a $\emptyset \subseteq Env$.

Osservazione 10: Variabili invalutabili

Dato un ambiente $E \in Env$, se $x \notin dom(E)$, ossia se x non è definita nell'ambiente E , allora x è una **variabile libera** e dunque è **invalutabile** in E , ossia:

$$\nexists v \in Val \text{ t.c. } E \vdash x \leadsto v$$

Esempio:

- L'espressione $x + 4$ è invalutabile, poiché $x \notin \text{dom}(\emptyset)$, dunque:

$$\nexists v' \in \text{Val t.c. } v = v' + 1 \wedge \frac{\emptyset \vdash x \rightsquigarrow v' \quad \emptyset \vdash 1 \rightsquigarrow 1}{\emptyset \vdash x + 1 \rightsquigarrow v}$$

- L'espressione $\text{let } x = 1 \text{ in } x + 4$ è valutabile, poiché $x \in \text{dom}(\{(x, 1)\})$, dunque:

$$\frac{\emptyset \vdash 1 \rightsquigarrow 1 \quad \frac{\{(x, 1)\} \vdash x \rightsquigarrow 1 \quad \{(x, 1)\} \vdash 4 \rightsquigarrow 4}{\{(x, 1)\} \vdash x + 1 \rightsquigarrow 5}}{\emptyset \vdash \text{let } x = 1 \text{ in } x + 4 \rightsquigarrow 5}$$

Definizione 23: Albero di derivazione

Definiamo come **albero di derivazione** l'albero generato dalla valutazione concatenata di più regole di inferenza.

Esempio:

- L'espressione $\text{let } y = 3 \text{ in } (\text{let } x = 7 \text{ in } x + y)$ viene valutata dal seguente albero di derivazione:

$$\frac{\emptyset \vdash 3 \rightsquigarrow 3 \quad \frac{\{y, 3\} \vdash 7 \rightsquigarrow 7 \quad \frac{\{(y, 3), (x, 7)\} \vdash x \rightsquigarrow 7 \quad \{(y, 3), (x, 7)\} \vdash y \rightsquigarrow 3}{\{(y, 3), (x, 7)\} \vdash x + y \rightsquigarrow 10}}{\{y, 3\} \vdash \text{let } x = 7 \text{ in } x + y \rightsquigarrow 10}}{\emptyset \vdash \text{let } y = 3 \text{ in } (\text{let } x = 7 \text{ in } x + y) \rightsquigarrow 10}$$

- Notiamo quindi come, per valutare l'intera espressione, ci basti in realtà valutare i termini "più in alto" dell'albero di derivazione

2.2 Valutazione Eager vs Lazy

Consideriamo la seguente espressione per il linguaggio *Exp*:

$$\text{let } x = \sqrt{397^5 + \int_3^{15} y^2 dy} + \log_{\sqrt{37}}(479) \text{ in } 3$$

Notiamo come nonostante l'espressione assegnata ad x sia di grandi dimensioni, richiedendo un enorme albero di derivazione, la valutazione dell'espressione sia totalmente indipendente da tale valutazione in quanto la variabile x non venga neanche utilizzata per la valutazione del secondo termine dell'espressione *let*.

Utilizzando le regole di valutazione previste dalla metodologia di valutazione, detta *eager* (trad: *affrettata*), vista nella sezione precedente, andremmo a valutare delle espressioni del tutto inutili.

Una metodologia di valutazione alternativa, detta *lazy*, è costituita da regole operazionali atte al *ritardare* la valutazione dei termini fino a quando non sia strettamente necessario.

Definizione 24: Valutazione eager

Definiamo una modalità di valutazione come **eager** (o valutazione *call-by-name* o *call-by-value*) se la valutazione di una sua espressione viene effettuata non appena essa viene legata ad una variabile, associandone immediatamente il risultato alla variabile stessa.

Definizione 25: Valutazione lazy

Definiamo una modalità di valutazione come **lazy** (o valutazione *call-by-need*) se la valutazione di una sua espressione viene effettuata solo quando si richiede il valore di un'espressione che da essa dipende.

Proposizione 7: Linguaggio *Exp* lazy

L'uso di una valutazione lazy necessita la ridefinizione dell'insieme *Env* e di alcune regole operazionali definite per la valutazione eager:

- L'insieme *Env* viene ridefinito come:

$$Env = \{f \mid f : Var \xrightarrow{fin} Exp\}$$

- Dato $E \in Env$, per le variabili si ha che:

$$x \in dom(E) \wedge E(x) = M \implies \frac{E \vdash M \rightsquigarrow v}{E \vdash x \rightsquigarrow v}$$

- Per l'espressione *let* si ha che:

$$\frac{E\{(x, M)\} \vdash N \rightsquigarrow v}{E \vdash let\ x = M\ in\ N \rightsquigarrow v}$$

Osservazione 11

È necessario puntualizzare che non sempre la valutazione lazy sia più ottimale della eager

Esempio:

- Consideriamo la seguente espressione

$$let\ x = M\ in\ x + x$$

- Utilizzando la valutazione eager otteniamo il seguente albero di derivazione:

$$\frac{\frac{\dots}{\emptyset \vdash M \rightsquigarrow v'} \quad \frac{\{(x, v')\} \vdash x \rightsquigarrow v' \quad \{(x, v')\} \vdash x \rightsquigarrow v'}{\{(x, v')\} \vdash x + x \rightsquigarrow v}}{\emptyset \vdash \text{let } x = M \text{ in } x + x \rightsquigarrow v}$$

dove $v = v' + v'$

- Utilizzando la valutazione lazy, invece, otteniamo il seguente albero di derivazione:

$$\frac{\frac{\dots}{\{(x, M)\} \vdash M \rightsquigarrow v'} \quad \frac{\{(x, M)\} \vdash M \rightsquigarrow v'}{\{(x, M)\} \vdash x \rightsquigarrow v'} \quad \frac{\dots}{\{(x, M)\} \vdash M \rightsquigarrow v'} \quad \frac{\{(x, M)\} \vdash M \rightsquigarrow v'}{\{(x, M)\} \vdash x \rightsquigarrow v'}}{\{(x, M)\} \vdash x + x \rightsquigarrow v} \quad \frac{\dots}{\emptyset \vdash \text{let } x = M \text{ in } x + x \rightsquigarrow v}$$

dove $v = v' + v'$

- Notiamo quindi che l'espressione M venga valutata una sola volta nella valutazione eager ma due volte nella valutazione lazy

2.3 Scoping Statico vs Dinamico

Consideriamo la seguente espressione:

$$\text{let } x = 3 \text{ in } (\text{let } y = x \text{ in } (\text{let } x = 7 \text{ in } y + x))$$

Prima di tutto, valutiamo tale espressione tramite valutazione eager:

$$\frac{\frac{\frac{\frac{\frac{\emptyset \vdash 3 \rightsquigarrow 3}{\{(x, 3)\} \vdash x \rightsquigarrow 3} \quad \frac{\frac{\frac{E \vdash 7 \rightsquigarrow 7 \quad E\{(x, 7)\} \vdash y \rightsquigarrow 3 \quad E\{(x, 7)\} \vdash x \rightsquigarrow 7}{E\{(x, 7)\} \vdash y + x \rightsquigarrow 10}}{\{(x, 3), (y, 3)\} \vdash \text{let } x = 7 \text{ in } y + x \rightsquigarrow 10}}{\{(x, 3)\} \vdash \text{let } y = x \text{ in } (\text{let } x = 7 \text{ in } y + x) \rightsquigarrow 10}}{\emptyset \vdash \text{let } x = 3 \text{ in } (\text{let } y = x \text{ in } (\text{let } x = 7 \text{ in } y + x)) \rightsquigarrow 10}}$$

dove $E := \{(x, 3), (y, 3)\}$

Valutiamo ora invece tale espressione utilizzando una valutazione lazy:

$$\frac{\frac{\frac{\frac{\frac{E\{(x, 7)\} \vdash 7 \rightsquigarrow 7}{E\{(x, 7)\} \vdash x \rightsquigarrow 7} \quad \frac{E\{(x, 7)\} \vdash 7 \rightsquigarrow 7}{E\{(x, 7)\} \vdash y \rightsquigarrow 7}}{\frac{E\{(x, 7)\} \vdash x \rightsquigarrow 7 \quad E\{(x, 7)\} \vdash y \rightsquigarrow 7}{E\{(x, 7)\} \vdash y + x \rightsquigarrow 14}}{\{(x, 3), (y, x)\} \vdash \text{let } x = 7 \text{ in } y + x \rightsquigarrow 14}}{\{(x, 3)\} \vdash \text{let } y = x \text{ in } (\text{let } x = 7 \text{ in } y + x) \rightsquigarrow 14}}{\emptyset \vdash \text{let } x = 3 \text{ in } (\text{let } y = x \text{ in } (\text{let } x = 7 \text{ in } y + x)) \rightsquigarrow 14}}$$

dove $E := \{(x, 3), (y, x)\}$

Notiamo quindi che le due valutazioni abbiano prodotto un risultato diverso. Tuttavia, vorremmo che le due valutazioni siano differenti solo a livello "*implementativo*", ossia che venga solo ritardata la valutazione dei termini. Difatti, tale problematica non è dovuta alla metodologia di valutazione utilizzata ma bensì dal tipo di *scoping*.

Definizione 26: Scoping statico

Definiamo un linguaggio come linguaggio a **scoping statico** se durante la valutazione di un'espressione viene utilizzato l'ambiente definito al tempo in cui viene interpretata (ma non valutata) l'espressione stessa.

Definizione 27: Scoping dinamico

Definiamo un linguaggio come linguaggio a **scoping dinamico** se durante la valutazione di un'espressione viene utilizzato l'ambiente definito al tempo di valutazione stesso.

Difatti, nell'esempio precedente ci troviamo in due situazioni:

- Nella valutazione eager, la variabile y viene valutata con l'ambiente $\{(x, 3), (y, x)\}$ definito al tempo in cui viene interpretata l'espressione $let\ y = x\ in\ \dots$ (scoping *statico*)
- Nella valutazione lazy, la variabile y viene valutata con l'ambiente $\{(x, 3), (y, x), (x, 7)\}$ definito al tempo della sua valutazione (scoping *dinamico*)

Per tanto, è necessario precisare che le due precedenti versioni viste del linguaggio *Exp* siano rispettivamente la versione **eager statica** e la versione **Exp lazy dinamica**.

Proposizione 8: Linguaggio *Exp* lazy statico

L'uso di una semantica lazy statica necessita la ridefinizione dell'insieme Env e di alcune regole operazionali definite per la semantica lazy dinamica:

- L'insieme Env viene ridefinito come:

$$Env = \{f \mid f : Var \xrightarrow{fin} Exp \times Env\}$$

- Dato $E \in Env$, per le variabili si ha che:

$$x \in dom(E) \wedge E(x) = (M, E') \implies \frac{E' \vdash M \rightsquigarrow v}{E \vdash x \rightsquigarrow v}$$

- Per l'espressione *let* si ha che:

$$\frac{E\{(x, (M, E))\} \vdash N \rightsquigarrow v}{E \vdash let\ x = M\ in\ N \rightsquigarrow v}$$

2.4 *Fun*: un linguaggio con funzioni

Definizione 29: Il linguaggio *Fun*

Definiamo come *Fun* il linguaggio rappresentato dalla seguente grammatica:

$$M, N ::= k \mid x \mid M + N \mid \text{let } x = M \text{ in } N \mid \text{fn } x \Rightarrow M \mid MN$$

dove:

- $k \in \{0, 1, \dots\}$ ossia è una **costante**
- $x \in \text{Var} = \{x, y, z, \dots\}$ ossia è una **variabile**
- $+$: $\text{Fun} \times \text{Fun} \rightarrow \text{Fun}$ la quale **somma le due espressioni**
- $\text{let} : \text{Var} \times \text{Fun} \times \text{Fun} \rightarrow \text{Fun}$ la quale **assegna** alla variabile x l'espressione M all'interno della **valutazione** di N . Inoltre, x prende il nome di variabile locale all'interno di N
- $\text{fn} : \text{Var} \times \text{Fun} \rightarrow \text{Fun}$ la quale restituisce una **funzione** avente un parametro il quale influenza l'espressione valutata dalla funzione
- Data l'espressione $\text{fn } x \Rightarrow M$, definiamo la coppia $(x, M) \in \text{Var} \times \text{Fun}$ come **chiusura** di tale espressione
- \cdot : $\text{Fun} \times \text{Fun} \rightarrow \text{Fun}$ la quale **applica** il termine sinistro al termine destro. In particolare, è necessario che il termine sinistro sia una funzione
- $\text{Val} = \{0, 1, \dots\} \cup (\text{Var} \times \text{Fun})$ è l'**insieme dei valori** in cui un'espressione può essere valutata, ossia costanti e chiusure

Esempi:

- L'espressione $(\text{fn } x \Rightarrow x + 1) \ 7$ viene valutata come 8, poiché la funzione sinistra $\text{fn } x \Rightarrow x + 1$ viene applicata al termine destro 7 (dunque 7 viene utilizzato come argomento della funzione per il parametro x)
- L'espressione $(\text{fn } x \Rightarrow x \ 3) \ 7$ è invalutabile, poiché l'argomento 7 viene passato come parametro x della funzione, ma all'interno di quest'ultima non è possibile valutare $x \ 3$ visto che 7 non è applicabile a 3
- L'espressione $(\text{fn } x \Rightarrow x \ 3)(\text{fn } x \Rightarrow x + 1)$ viene valutata come 4, poiché l'argomento $\text{fn } x \Rightarrow x + 1$ viene passato come parametro x della funzione $\text{fn } x \Rightarrow x \ 3$, per poi valutare l'applicazione $x \ 3$ passando l'argomento 3 come parametro per la funzione contenuta in x (ossia $\text{fn } x \Rightarrow x + 1$).

Informalmente, possiamo dire che:

$$(\text{fn } x \Rightarrow x \ 3)(\text{fn } x \Rightarrow x + 1) \longrightarrow (\text{fn } x \Rightarrow x + 1) \ 3 \longrightarrow 4$$

Osservazione 14

Nel caso in cui si abbia un'espressione con doppio operatore di applicazione MNL , essa verrà valutata come $(MN)L$

Esempio:

- Le due espressioni $(fn\ x \Rightarrow x\ 3)(fn\ x \Rightarrow x + 1)\ 7$ e $[(fn\ x \Rightarrow x\ 3)(fn\ x \Rightarrow x + 1)]\ 7$ sono equivalenti

Definizione 30: Curryficazione

Definiamo come **curryficazione** la contrazione sintattica $fn\ x_1 x_2 \dots x_n \Rightarrow M$ equivalente alla seguente espressione:

$$fn\ x_1 \Rightarrow (fn\ x_2 \Rightarrow \dots (fn\ x_n \Rightarrow M) \dots)$$

Esempi:

- L'uncurryficazione dell'espressione $(fn\ xy \Rightarrow yx)\ 7\ (fn\ x \Rightarrow x + 1)$ corrisponde a:

$$(fn\ x \Rightarrow fn\ y \Rightarrow yx)\ 7\ (fn\ x \Rightarrow x + 1)$$

e viene pertanto valutata come 8. Difatti, informalmente, possiamo dire che:

$$(fn\ x \Rightarrow fn\ y \Rightarrow yx)\ 7\ (fn\ x \Rightarrow x + 1) \longrightarrow (fn\ y \Rightarrow y\ 7)(fn\ x \Rightarrow x + 1) \longrightarrow 8$$

Osservazione 15

Trattandosi di un'estensione del linguaggio *Exp*, il linguaggio *Fun* **eredita le regole operazionali** delle semantiche di *Exp*

Proposizione 9: Linguaggio *Fun* eager dinamico

La semantica eager dinamica del linguaggio *Fun* prevede l'aggiunta di alcune regole operazionali:

- L'insieme *Env* viene ridefinito come:

$$Env = \{f \mid f : Var \xrightarrow{fin} Val\}$$

- Dato $E \in Env$, per le funzioni si ha che:

$$E \vdash fn\ x \Rightarrow M \rightsquigarrow (x, M)$$

- Dato $E \in Env$, per le applicazioni si ha che:

$$\frac{E \vdash M \rightsquigarrow (x, L) \quad E \vdash N \rightsquigarrow v' \quad E\{(x, v')\} \vdash L \rightsquigarrow v}{E \vdash MN \rightsquigarrow v}$$

Proposizione 10: Linguaggio *Fun* eager statico

La semantica eager statica del linguaggio *Fun* prevede l'aggiunta di alcune regole operazionali:

- L'insieme *Env* viene ridefinito come:

$$Env = \{f \mid f : Var \xrightarrow{fin} Val \times Env\}$$

- Dato $E \in Env$, per le funzioni si ha che:

$$E \vdash fn\ x \Rightarrow M \rightsquigarrow (x, M, E)$$

- Dato $E \in Env$, per le applicazioni si ha che:

$$\frac{E \vdash M \rightsquigarrow (x, L, E') \quad E \vdash N \rightsquigarrow v' \quad E'\{(x, v')\} \vdash L \rightsquigarrow v}{E \vdash MN \rightsquigarrow v}$$

Lemma 2

A differenza del linguaggio *Exp*, per la sua estensione *Fun* si ha che:

$$Fun\ \text{eager}\ \text{dinamico} \not\equiv Fun\ \text{eager}\ \text{statico}$$

Dimostrazione.

- Consideriamo l'espressione $let\ x = 7\ in\ ((fn\ y \Rightarrow let\ x = 3\ in\ yx)(fn\ z \Rightarrow x))$
- Utilizzando la semantica eager dinamica, l'albero di derivazione corrisponde a:

$$\begin{array}{c}
 (*) \quad \frac{E' \vdash 3 \rightsquigarrow 3 \quad \frac{E'' \vdash y \rightsquigarrow (z, x) \quad E'' \vdash x \rightsquigarrow 3 \quad E''\{(z, 3)\} \vdash x \rightsquigarrow 3}{E'' \vdash yx \rightsquigarrow 3}}{E' \vdash let\ x = 3\ in\ yx \rightsquigarrow 3} \\
 \\
 \frac{\emptyset \vdash 7 \rightsquigarrow 7 \quad \frac{E \vdash fn\ y \Rightarrow let\ x = 3\ in\ yx \rightsquigarrow (y, let\ x = 3\ in\ yx) \quad E \vdash fn\ z \Rightarrow x \rightsquigarrow (z, x) \quad (*)}{E \vdash (fn\ y \Rightarrow let\ x = 3\ in\ yx)(fn\ z \Rightarrow x) \rightsquigarrow 3}}{
 \emptyset \vdash let\ x = 7\ in\ ((fn\ y \Rightarrow let\ x = 3\ in\ yx)(fn\ z \Rightarrow x)) \rightsquigarrow 3}
 \end{array}$$

dove $E := \{(x, 7)\}$, $E' := E\{(y, \{(z, x)\})\}$ e $E'' := E'\{(x, 3)\}$

- Utilizzando la semantica eager statica, invece, l'albero di derivazione corrisponde a:

$$(*) \quad \frac{E' \vdash 3 \rightsquigarrow 3 \quad \frac{E'' \vdash y \rightsquigarrow (z, x, E) \quad E'' \vdash x \rightsquigarrow 3 \quad E\{(z, 3)\} \vdash x \rightsquigarrow 7}{E'' \vdash yx \rightsquigarrow 7}}{E' \vdash let\ x = 3\ in\ yx \rightsquigarrow 7}$$

$$\frac{\emptyset \vdash 7 \rightsquigarrow 7 \quad \frac{E \vdash fn\ y \Rightarrow let\ x = 3\ in\ yx \rightsquigarrow (y, let\ x = 3\ in\ yx, E) \quad E \vdash fn\ z \Rightarrow x \rightsquigarrow (z, x, E) \quad (*)}{E \vdash (fn\ y \Rightarrow let\ x = 3\ in\ yx)(fn\ z \Rightarrow x) \rightsquigarrow 7}}{\emptyset \vdash let\ x = 7\ in\ ((fn\ y \Rightarrow let\ x = 3\ in\ yx)(fn\ z \Rightarrow x)) \rightsquigarrow 7}$$

dove $E := \{(x, 7)\}$, $E' := E\{(y, \{(z, x, E)\})\}$ e $E'' := E'\{(x, 7)\}$

- Poiché l'espressione restituisce due valutazioni diverse, le due semantiche non sono equivalenti

□

Proposizione 11: Linguaggio *Fun* lazy dinamico

La semantica lazy dinamica del linguaggio *Fun* prevede l'aggiunta di alcune regole operazionali:

- L'insieme Env viene ridefinito come:

$$Env = \{f \mid f : Var \xrightarrow{fin} Fun\}$$

- Dato $E \in Env$, per le funzioni si ha che:

$$E \vdash fn\ x \Rightarrow M \rightsquigarrow (x, M)$$

- Dato $E \in Env$, per le applicazioni si ha che:

$$\frac{E \vdash M \rightsquigarrow (x, L) \quad E'\{(x, N)\} \vdash L \rightsquigarrow v}{E \vdash MN \rightsquigarrow v}$$

Proposizione 12: Linguaggio *Fun* lazy statico

La semantica lazy statica del linguaggio *Fun* prevede l'aggiunta di alcune regole operazionali:

- L'insieme Env viene ridefinito come:

$$Env = \{f \mid f : Var \xrightarrow{fin} Fun \times Env\}$$

- Dato $E \in Env$, per le funzioni si ha che:

$$E \vdash fn\ x \Rightarrow M \rightsquigarrow (x, M)$$

- Dato $E \in Env$, per le applicazioni si ha che:

$$\frac{E \vdash M \rightsquigarrow (x, L, E') \quad E'\{(x, N, E)\} \vdash L \rightsquigarrow v}{E \vdash MN \rightsquigarrow v}$$

Osservazione 16

Come per il linguaggio *Exp*, per la sua estensione *Fun* si ha che:

$$Fun \text{ lazy dinamico} \not\equiv Fun \text{ lazy statico}$$

Definizione 31: Espressione ω

Dato il linguaggio *Fun*, definiamo come **espressione omega**, indicata con ω , la seguente espressione:

$$\omega := (fn\ x \Rightarrow xx)(fn\ x \Rightarrow xx)$$

In particolare, l'espressione ω è **invalutabile per qualsiasi semantica**

Esempio:

- Analizziamo l'albero di derivazione di ω utilizzando una semantica eager statica:

$$\begin{array}{c}
 (*) \quad \emptyset \vdash x \rightsquigarrow (x, xx, \emptyset) \quad \emptyset \vdash x \rightsquigarrow (x, xx, \emptyset) \quad \frac{(*)}{(x, \{(x, xx, \emptyset)\}) \vdash xx \rightsquigarrow v} \\
 \hline
 \emptyset \vdash fn\ x \Rightarrow xx \rightsquigarrow (x, xx, \emptyset) \quad \emptyset \vdash fn\ x \Rightarrow xx \rightsquigarrow (x, xx, \emptyset) \quad \frac{(*)}{(x, \{(x, xx, \emptyset)\}) \vdash xx \rightsquigarrow v} \\
 \hline
 \emptyset \vdash (fn\ x \Rightarrow xx)(fn\ x \Rightarrow xx) \rightsquigarrow v
 \end{array}$$

- Notiamo quindi che affinché la valutazione del termine $(x, \{(x, xx, \emptyset)\}) \vdash xx \rightsquigarrow v$ richieda che esso stesso venga valutato, creando così un albero di valutazione infinito.

Lemma 3

Dato il linguaggio *Fun*, si ha che:

$$Fun \text{ eager statico} \not\equiv Fun \text{ lazy statico}$$

$$Fun \text{ eager dinamico} \not\equiv Fun \text{ lazy dinamico}$$

Dimostrazione.

- Consideriamo l'espressione $let\ x = \omega\ in\ 42$. Utilizzando una semantica eager (statica o dinamica), verrebbe richiesta immediatamente la valutazione del termine ω , il quale tuttavia è invalutabile. Utilizzando una semantica lazy (statica o dinamica), invece, il termine ω non verrà mai valutato, restituendo 42 come risultato.

□

Teorema 5: Equivalenze semantiche di *Fun*

Dato il linguaggio *Fun*, **non esistono due semantiche equivalenti**