

Linguaggi di programmazione

Numeri naturali

In base a chi lo si chiede si avranno risposte diverse sul dare una definizione di numeri naturali

es

$3 = \{\{3\}, \{\{3\}\}, \{\{\{3\}\}\}, \{\{\{\{3\}\}\}\}\}$ Von Neumann → definizione insiemistica

$3 = \lambda xy.x(x(y))$ Alonzo Church → definizione ricorsiva

Immaginiamo di parlare con un marziano e cerchiamogli di dare una definizione di successore nei numeri naturali (lui tenterà di smentirlo, quindi bisogna fare attenzione) secondo gli assiomi di Peano

1. $0 \in \mathbb{N}$

2. $n \in \mathbb{N} \rightarrow \text{succ}(n) \in \mathbb{N}$

3. $\nexists n \text{ t.c. } 0 = \text{succ}(n)$

4. $\forall n, m \text{ } \text{succ}(n) = \text{succ}(m) \Rightarrow n = m$

5. $\forall S \subseteq \mathbb{N} \exists 0 \in S$

$\{n \in S \Rightarrow \text{succ}(n) \in S\}$

$\Rightarrow S = \mathbb{N}$ → per evitare di equiparare \mathbb{N} ad insiemi che rispettano la struttura di \mathbb{N} ma contengono anche qualcosa in più

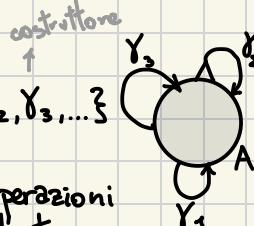
una proprietà e il sottoinsieme corrispondente sono in biezione (dal sottoinsieme posso ricavare la proprietà originale)

induzione

$$\begin{array}{c} \forall P \quad P(0) \quad P(n) \rightarrow P(n+1) \\ \hline \forall n \quad P(n) \end{array}$$

Definire un'algebra

(A, Γ) dove $\Gamma = \{\gamma_1, \gamma_2, \gamma_3, \dots\}$



Ma posso anche definire operazioni che prendono elementi dall'esterno

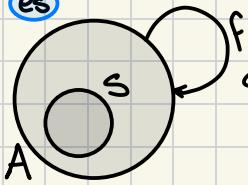
Chiusura di un'algebra

Sia $f: A^n \times K \rightarrow A$ una operazione su A con parametri esterni in $K = K_1 \times \dots \times K_m$.

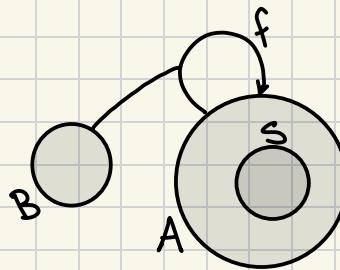
Un insieme $S \subseteq A$ si dice **chiuso** rispetto a f quando $a_1, \dots, a_n \in S \rightarrow f(a_1, \dots, a_n, K_1, \dots, K_m) \in S$

Ovvero è chiuso se per ogni input possibile che pesco da A e che proviene da S , questo rimane in f (quelli che provengono da $A \setminus S$ o dall'esterno non ci interessano)

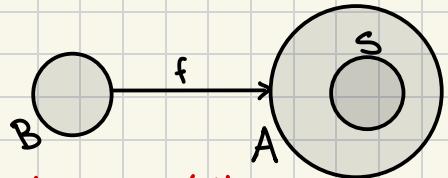
es



S è chiuso rispetto a f se
 $\forall x \in S \rightarrow f(x) \in S$



S è chiuso rispetto a f se
 $\forall x \in B, y \in S \rightarrow f(x, y) \in S$



S è chiuso rispetto a f se
 $\forall x \in B \rightarrow f(x) \in S$

Algebra induttiva

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

- tutte le γ_i sono iniettive

un elemento non può essere immagine

- tutte le γ_i sono disgiunte

di due funzioni differenti

- $\forall S \subseteq A$ se S è chiuso rispetto a tutte le γ_i allora $S = A$ → principio di induzione

applicando la def vedremo che, visto che possiamo usare qualsiasi elemento di S , la seconda parte dell'implicazione (visto che prende input solo da B) è vera se per qualsiasi input di f finisce in S

es IN

in \mathbb{N} però abbiamo parlato dell'elemento 0 (ora non contemplato), definiamo quindi la seguente algebrā

$(\mathbb{N}, \{\emptyset, \text{succ}\})$ dove $\text{succ} : \mathbb{N} \rightarrow \mathbb{N}$, $\mathbb{N} \xrightarrow{\text{succ}} \mathbb{N}$

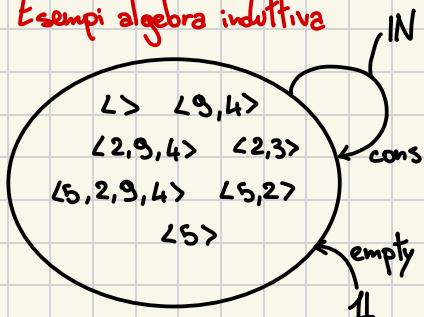
funzione che associa un insieme di un solo elemento all'elemento 0 in \mathbb{N}

Notiamo quindi che:

- tutte le γ_i sono iniettive.
- tutte le immagini di γ_i sono disgiunte (\emptyset punta a 0, mentre succ a tutti gli altri)
- \mathbb{N} è chiuso rispetto a succ
- \mathbb{N} è chiuso rispetto a \emptyset

Il costruire un'algebra induttiva mi garantisce che non esistono sottosalgabre in essa contenute. Inoltre tutte le algebre induttive sono **isomorfismi**.

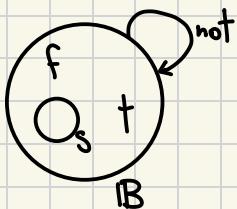
Esempi algebra induttiva



$\text{cons}(2, \langle 3, 4 \rangle) = \langle 2, 3, 4 \rangle \leftarrow \text{con append non sarebbe stata induttiva, non disgiunta}$

L'insieme che contiene le liste finite è induttivo, mentre quello che contiene quelle finite e infinite poiché contiene l'algebra delle liste finite, non lo è.

esistono algebre induttive senza operazioni nullarie? exp.



L'insieme $S = \emptyset$ è chiuso poiché non ci sono elementi in S su cui poter applicare la funzione, la prima parte dell'implicazione della def. di chiusura è falsa, rimane solo da vedere se l'output è.

In particolare l'insieme vuoto non è chiuso per nessuna operazione nullaria ma lo è per tutte le operazioni con arità ≥ 1 .

1. Costruttori "Nullari" (Costanti, Arità = 0)

- Esempio: `empty`, `0`, `true`.
- Verdetto: L'insieme vuoto NON è chiuso.
- Perché: Come abbiamo visto, la definizione impone che la costante appartenga all'insieme ($c \in S$). Poiché \emptyset non contiene nulla, questa condizione fallisce sempre.
- Quindi: Hai ragione solo per questo tipo di costruttori.

Tuttavia, un'operazione nullaria ($n = 0$) è, di fatto, una **costante**. Non prende input. La definizione di chiusura per una nullaria c è semplicemente:

$$c \in S$$

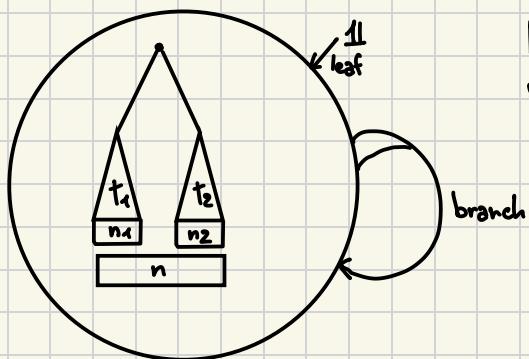
2. Costruttori "Non Nullari" (Arità ≥ 1)

- Esempio: `cons(x, list)`, `successor(n)`, `sum(a, b)`.
- Verdetto: L'insieme vuoto È chiuso.
- Perché: Qui vale la "verità a vuoto" (vacuous truth). La definizione dice: "Se prendo elementi da \emptyset , il risultato è in \emptyset ". Poiché non posso prendere elementi da \emptyset , la premessa è falsa e l'intera affermazione logica risulta VERA.
- In parole povere: Non puoi "rompere" la chiusura se non hai materiali con cui costruire qualcosa che porti fuori dall'insieme.

TEOREMA

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

Per dimostrarlo dovranno usare l'induzione strutturale



Nell'induzione strutturale assumiamo che la proprietà sia valida per t_1 e t_2

$$\begin{aligned} n &= n_1 + n_2 \\ (2n_1 - 1) + (2n_2 - 1) + 1 &= 2(n_1 + n_2) - 1 \end{aligned}$$

radice

\downarrow

n

Espresione link

Definiamo un linguaggio L come insieme di stringhe. Per descrivere la sintassi di linguaggi formali, usiamo la seguente sintassi:

$\langle \text{simbolo} \rangle ::= \langle \text{espressione} \rangle$

es

$M, N ::= s \mid ? \mid M+N \mid M*N$

$\underline{\quad} \quad \underline{\quad} \quad \underline{\quad}$ assegnazione "secca"

dove se M e N sono espressioni è possibile costruire un'altra espressione unendoli tramite "+" o "*"

Definiamo quindi una funzione $\text{eval}: \text{Exp} \rightarrow \text{IN}$

es elemento di Exp

$\text{eval}(s) = s$ elemento di IN

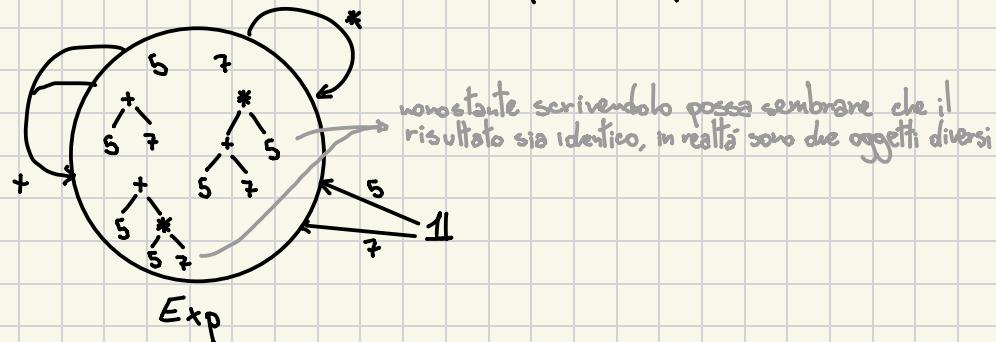
$\text{eval}(?) = ?$

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

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

$\text{eval}(\boxed{s+?*s}) = ?$

infatti in base a come considero l'espressione posso avere risultati diversi, dobbiamo quindi **disambiguare**



nonostante scrivendolo possa sembrare che il risultato sia identico, in realtà sono due oggetti diversi

sicuramente non si tratta di una grammatica induttiva, ma ho usato l'induzione per definirla

Equazione ricorsive di domini

es

$$\begin{aligned} \mathbb{N} &= 1\mathbb{N} + \mathbb{N} \\ L &= \mathbb{N} \cup (\mathbb{N} \times L) \end{aligned}$$

isomorfismo 2
insieme di liste

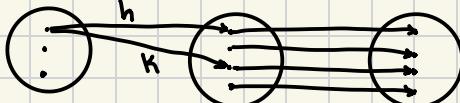
Brief sulla teoria delle categorie

Nella teoria delle categorie non si può usare la parola "elemento"

Come faccio quindi a descrivere un morfismo (funzione) f iniettivo (monomorfismo) non potendo parlare di elementi?

RIS $h \circ f = h \circ f \Rightarrow h = h$

$$A \xrightarrow{h} B \xrightarrow{f} C$$



essendo f iniettiva ad ogni elemento viene associato un elemento diverso del codominio, dunque nella composizione $h \circ f$ e $h \circ f$ risulta che gli elementi di destinazione sono esattamente gli stessi allora $h = h$

Segnatura di un'algebra

Due algebre (A, Γ_A) , (B, Γ_B) hanno la stessa segnatura se $\forall \gamma_A \in \Gamma_A \exists \gamma_B \in \Gamma_B$ con stessa aritmetà e se, sostituendo tutte le B in γ_B con A ottengo γ_A

es

algebra D

$$f_D: A \times D \rightarrow D$$

$$g_D: 1\mathbb{N} \rightarrow D$$

$$h_D: A \times B \times D \rightarrow D$$

algebra di C

$$f_C: A \times C \rightarrow C$$

$$g_C: 1\mathbb{N} \rightarrow C$$

$$h_C: A \times B \times C \rightarrow C$$

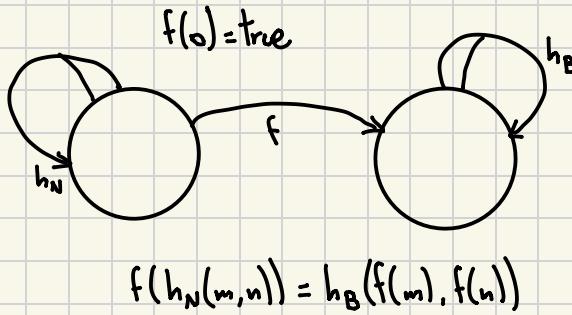
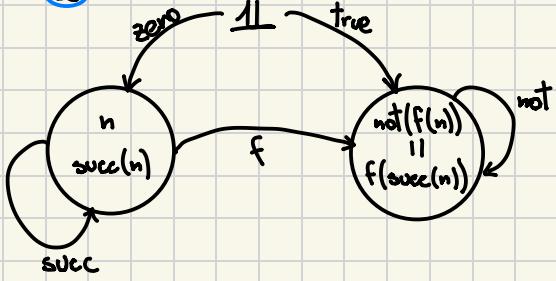
D e C hanno la stessa segnatura

Omomorfismo

Due algebre con la stessa segnatura (A, Γ) e $(B, \Delta = \{\delta_1, \dots, \delta_K\})$, un omomorfismo è una funzione $A \rightarrow B$ tale che:

$$\forall i: f(\gamma_i(a_1, \dots, a_n)) = \delta_i(f(a_1), \dots, f(a_n)) \quad \text{rispetta le operazioni}$$

es



$$f(h_N(m, n)) = h_B(f(m), f(n))$$

Se applico l'operazione da una parte o dall'altra il risultato non cambia, e lo devo fare per tutte le operazioni dell'algebra

TEOREMA

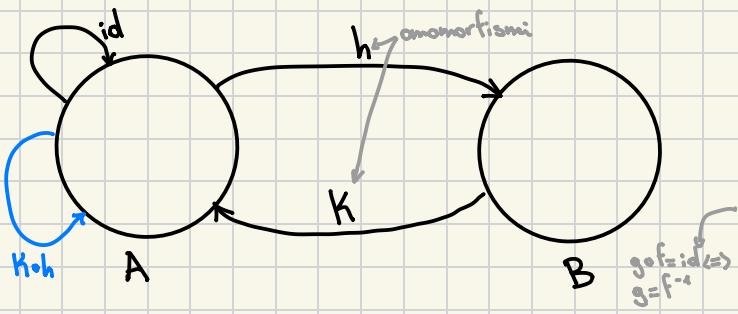
Prese due algebre induttive che hanno la stessa segnatura allora esiste un unico omomorfismo

DEF Isomorfismo

Un isomorfismo è un omomorfismo in cui la funzione è una biezione

LEMMA di Lambek

Avendo due algebre induttive con stessa segnatura allora sono necessariamente isomorfe



componendo h e K otteniamo dunque, per quanto detto, un ulteriore omomorfismo, ma in A già esiste un omomorfismo ovvero l'identità. Per il teorema precedente però può esistere un solo omomorfismo dunque $id = K \circ h : A \rightarrow A$
da ciò si deduce che K e h sono inversi per cui la loro composizione è chiaramente biettiva, il che ci fa concludere che $K \circ h$ è un isomorfismo, quindi $A \cong B$

Linguaggio Exp link

es

$M, N, L ::= K | x | M+N | \text{let } x = M \text{ in } N$ ↗ ML Standard

dove:

- $K \rightarrow \text{costante}$

- $x \rightarrow \text{variabile } \in \text{Var}$

- $\text{let } x = M \text{ in } N \rightarrow \text{assegne a } x \text{ il valore } M \text{ all'interno di } N$

Vediamo alcuni esempi di uso:

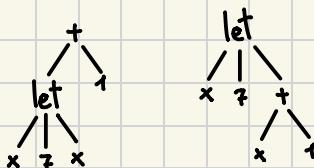
$\text{let } x = 3 \text{ in } x+x+2 \Rightarrow 8$

$\text{let } x = x+2 \text{ in } 4 \Rightarrow 4$

$\text{let } x = \text{let } y = 3 \text{ in } y+1 \text{ in } x+y \Rightarrow 7$

$\text{let } x = 7 \text{ in } x+1 \Rightarrow ??$

quest'ultima espressione può creare ambiguità
dato che può essere doppiamente interpretata



DEF Variabili libere e legate

Una variabile x si dice **libera** se data un'exp $\text{let } x = M \text{ in } N$, x non compare in N

Si dice invece **legata** se invece compare in N

es

$\text{let } x = 5 \text{ in } x+x$
legata
libera

DEF free

Definiamo la funzione free: $\text{Exp} \rightarrow \wp(\text{Var})$ che restituisce l'insieme delle variabili libere di un'espressione

es

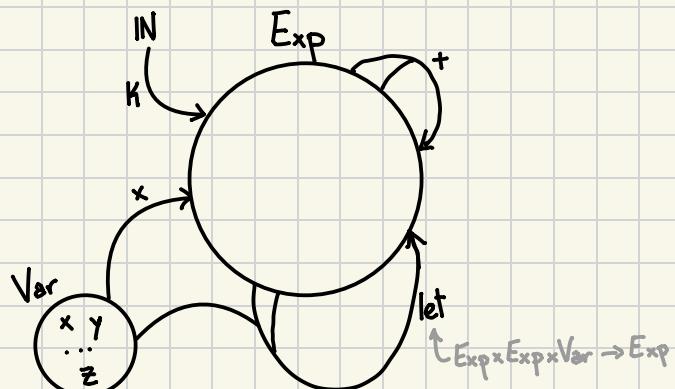
$\text{free}(\text{let } x = 7 \text{ in } x+y) = \{y\}$

$\text{free}(K) = \emptyset$

$\text{free}(x) = \{x\}$

$\text{free}(M+N) = \text{free}(N) \cup \text{free}(M)$

$\text{free}(\text{let } x = M \text{ in } N) = \text{free}(M) \cup (\text{free}(N) - x) \leftarrow \text{free}(\text{let } x = x \text{ in } x) = x$



$\uparrow \text{Exp} \times \text{Exp} \times \text{Var} \rightarrow \text{Exp}$

Semantica operazionale

La semantica operazionale permette di rispondere alla domanda quanto fa, permettendoci di eliminare le ambiguità viste finora

Definisco quindi un'operazione per calcolare il valore, ma dipende anche dall'**ambiente** che si sta considerando

$$M \in \text{Exp} \times \text{Env} \rightsquigarrow \text{Val}$$

dove $\rightsquigarrow \subseteq \text{Exp} \times \text{Exp} \times \text{Val}$

↑
associazione tra
variabile e valore

↑
ambiente in cui
valutare Exp

Poiché ci sono casi in cui non viene restituito alcun risultato, si tratta quindi di una **relazione** e non di una funzione

Mi sto quindi chiedendo se $(M, E, v) \in \rightsquigarrow$

questa espressione ha questo valore

es)

$$(\text{let } x=3 \text{ in } x+1, [x \mapsto 7], 10) \in \rightsquigarrow$$

$$(s, [E], 7) \notin \rightsquigarrow$$

Passiamo ora a una notazione più snella

deduzione

$$\boxed{E \vdash M \rightsquigarrow v}$$

↓ sequenze

che si legge "M deduce E con valore v"

deduzione:

$$A, A \rightarrow B \vdash B$$

deduco che
/ ho l'ombrello

pioggia → se pioggia porta
l'ombrello

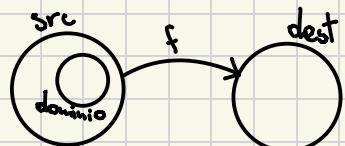
conseguenza logica ≠ deduzione

Ambiente

L'ambiente è una funzione parziale a dominio finito da variabile a valore

→ dominio finito

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



$$f: \text{src} \longrightarrow \text{dest}$$

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

es)

$$E_1 = \{(z, 3), (y, 9)\}$$

$$E_2 = \{(z, 4)\}$$

$$(E_1, E_2)(y) = 9$$

$$(E_1, E_2)(z) = 4$$

Regola di inferenza eager

- ① $E \vdash K \rightsquigarrow K$ const
 ② $E \vdash x \rightsquigarrow E(x)$ x non esiste non esiste nessuna triple var
 ↗ quando fa x
 nell'ambiente E ?

- $$\textcircled{3} \quad \begin{array}{c} E \vdash M \rightsquigarrow v_1 & E \vdash N \rightsquigarrow v_2 \\ \hline E \vdash M + N \rightsquigarrow v \end{array} \quad (\text{dove } v = v_1 + v_2) \quad \begin{array}{l} \text{condizioni laterali} \\ \text{plus} \end{array}$$

conclusione

- $$\begin{array}{c} \textcircled{4} \\ \text{valuta M con E} \\ \text{e mettilo in } v \\ \text{E} \vdash M \rightsquigarrow v \\ \hline \text{E} \vdash \text{let } x = M \text{ in } N \rightsquigarrow v' \end{array} \quad \begin{array}{c} \text{valuta N con E insieme a} \\ (x, v) \text{ e mettilo in } v' \\ \text{E} \{ (x, v) \} \vdash N \rightsquigarrow v' \\ \text{let} \end{array}$$

es $\emptyset \vdash \text{let } x = 3 \text{ in } ((\text{let } x = (\text{let } y = 2 \text{ in } x+y) \text{ in } x+7) + x)$
 adesso applichiamo le regole di inferenza dall'esterno all'interno ↗
 per prima uso quella del let, spostando l'ultima parentesi se rebbe stata quella del +

$\frac{(x,3)(y,2) \vdash x \rightsquigarrow 3 \quad (x,3)(y,2) \vdash y \rightsquigarrow 2}{(x,3) \vdash 2 \rightsquigarrow 2 \quad (x,3)(y,2) \vdash x+y \rightsquigarrow 5}$ plus
 $\text{let } y=2 \text{ in } x+y \rightsquigarrow 5$
 $\frac{(x,3) \vdash \text{let } y=2 \text{ in } x+y \rightsquigarrow 5 \quad (x,3)(x,5) \vdash x+7 \rightsquigarrow 12}{(x,3) \vdash \text{let } x=(\text{let } y=2 \text{ in } x+y) \text{ in } x+7 \rightsquigarrow 12}$ let
 $\text{let } x=(\text{let } y=2 \text{ in } x+y) \text{ in } x+7 \rightsquigarrow 12$
 $(x,3) \vdash x \rightsquigarrow 3$ plus
 $\emptyset \vdash 3 \rightsquigarrow 3 \quad \{(x,3)\} \vdash (\text{let } x=(\text{let } y=2 \text{ in } x+y) \text{ in } x+7) + x \rightsquigarrow 15$
 $\emptyset \vdash \text{let } x=3 \text{ in } ((\text{let } x=(\text{let } y=2 \text{ in } x+y) \text{ in } x+7) + x) \rightsquigarrow 15$

L'approccio eager però ha uno svantaggio, infatti in un caso del tipo `let x = f()` ~~f()~~ viene valutato prima il "mostro" nonostante in questo caso non sia assolutamente necessario. Come soluzione viene usato l'approccio lazy.

Semantica lazy-dinamica

Rispetto alle regole di inferenza eager cambiano quella del let e quella della variabile

- $$\textcircled{1} \quad \frac{E \vdash M \rightsquigarrow J}{E \vdash x \rightsquigarrow J} (\text{se } Ex = M) \quad \text{var}$$

oss $\emptyset \vdash \text{let } x = y \text{ in let } y = z \text{ in } x$ in lazy dinamico
fa z ma dovrebbe essere non definito

- $$\textcircled{2} \quad \frac{E(x, M) \vdash N \rightsquigarrow v}{F \vdash \text{let } x = M \text{ in } N \rightsquigarrow v} \text{ let}$$

⑤ let $x=2$ in let $y=x+1$ in let $x=7$ in y

eager

$$\frac{(x,2) \vdash x \rightsquigarrow 2 \quad (x,2) \vdash 1 \rightsquigarrow 1}{(x,2) \vdash x+1 \rightsquigarrow 3} \quad \frac{(x,2)(y,3) \vdash ? \rightsquigarrow ? \quad (x,2)(y,3)(x,?) \vdash y \rightsquigarrow 3}{(x,2)(y,3) \vdash \text{let } x=? \text{ in } y \rightsquigarrow 3}$$

$\emptyset \vdash 2 \rightsquigarrow 2$ $(x, 2) \vdash \text{let } y = x + 1 \text{ in let } x = 7 \text{ in } y \rightsquigarrow 3$

$\emptyset \vdash \text{let } x=2 \text{ in let } y=x+1 \text{ in let } x=7 \text{ in } y \rightsquigarrow 3$

lazy-dinamico

$$\begin{array}{l}
 \frac{(x, z)(y, x+1)(x, z) \vdash x \rightsquigarrow z}{(x, z)(y, x+1)(x, z) \vdash 1 \rightsquigarrow 1} \\
 \frac{(x, z)(y, x+1)(x, z) \vdash x+1 \rightsquigarrow 8}{(x, z)(y, x+1)(x, z) \vdash y \rightsquigarrow 8} \\
 \frac{(x, z)(y, x+1) \vdash \text{let } x = ? \text{ in } y \rightsquigarrow 8}{\emptyset (x, z) \vdash \text{let } y = x+1 \text{ in let } x = ? \text{ in } y \rightsquigarrow 8}
 \end{array}$$

diverso

Introducendo il lazy ci siamo esposti al problema dello scoping (in questo caso è dinamico). La semantica lazy-dinamica è considerata "sbagliata" poiché ci dà un risultato diverso dalle altre 3 combinazioni di metodologie.

Semantica lazy-statica

Rispetto alle regole di inferenza eager cambiano quella del let e quella della variabile.

①

$$\frac{E \vdash M \rightsquigarrow v}{E' \vdash x \rightsquigarrow v} \quad (\text{se } E'(x) = (M, E)) \quad \text{var}$$

②

$$\frac{E(x, M, E) \vdash N \rightsquigarrow v}{E \vdash \text{let } x = M \text{ in } N \rightsquigarrow v} \quad \text{let} \quad E : \text{Var} \xrightarrow{\text{fin}} \text{Exp} \times \text{Env}$$

OSS in questo caso let $x = \dots$ in $x+x+x+x+x$ la semantica lazy è molto lenta (devo ricalcolare ogni volta x)

es

$$\begin{array}{c} \frac{\emptyset \vdash 2 \rightsquigarrow 2}{(x, 2, \emptyset) \vdash x \rightsquigarrow 2} \quad \frac{(x, 2, \emptyset) \vdash 1 \rightsquigarrow 1}{(x, 2, \emptyset) \vdash x+1 \rightsquigarrow 3} \\ \hline \frac{}{E(x, 7, E) \vdash y \rightsquigarrow 3} \\ \hline \frac{E \leftarrow (x, 2, \emptyset)(y, x+1, (x, 2, \emptyset)) \text{ let } x=7 \text{ in } y \rightsquigarrow 3}{(x, 2, \emptyset) \vdash \text{let } y = x+1 \text{ in let } x=7 \text{ in } y \rightsquigarrow 3} \\ \hline \emptyset \vdash \text{let } x=2 \text{ in let } y = x+1 \text{ in let } x=7 \text{ in } y \rightsquigarrow 3 \end{array}$$

TEOREMA

Semantica eager e lazy-statica sono equivalenti.

	statico	dinamico
lazy	○	✗
eager	○	○

il linguaggio Exp è troppo semplice per poter cogliere la differenza tra eager statico e dinamico

Linguaggio Fun
 $M, N ::= S \mid x \mid M+N \mid \text{let } x = M \text{ in } N \mid f_n \ x \Rightarrow M \mid MN$

es

$$(f_n \ x \Rightarrow x+1) \ S \rightsquigarrow 6$$

M N

↳ funzione M applicata a argomento N

$$(f_n \ x \Rightarrow x+1)(f_n \ z \Rightarrow z)$$

restituisce la funzione alla quale
se passo in input una funzione
restituisce una funzione

$$[(f_n \ x \Rightarrow [f_n \ y \Rightarrow y(x+1)]) \ S_2] (f_n \ z \Rightarrow z+1) \rightsquigarrow S_4$$

z \Rightarrow z+1 S_2

si possono non solo avere valori in output ma anche intere funzioni
 $(f_n \ z \Rightarrow z)(f_n \ x \Rightarrow x+1) \rightsquigarrow f_n \ x \Rightarrow x+1$

oss esistono termini che non hanno una semantica (leggibili da scrivere ma che non danno alcun risultato)
come ad esempio $S(f_n \ x \Rightarrow x+1)$

oss Fun non è né commutativo né associativo

Semantica eager dinamica

Inoltre a differenza di Exp, Env: var $\xrightarrow{\text{fun}}$ val dove val = {5, 42, ...} \cup (var \times Fun)

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

chiatura

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

$$f_n \ x \Rightarrow x+1 \rightarrow \overbrace{(x, x+1)}^{\text{chiusura}}$$

es

$$\frac{\begin{array}{c} E' \vdash y \rightsquigarrow (z, 3) \quad E' \vdash s \rightsquigarrow E'(z, 5) \vdash 3 \rightsquigarrow 3 \\ E \vdash f_n \ y \Rightarrow y \ S \rightsquigarrow (y, y5) \quad E \vdash f_n \ z \Rightarrow 3 \rightsquigarrow (z, 3) \quad E' \vdash (y, (z, 3)) \vdash y5 \rightsquigarrow 3 \quad E(x, 3) \vdash x \rightsquigarrow 3 \quad E(x, 3) \vdash x+1 \rightsquigarrow 4 \\ (z, (x, x+1)) \vdash z \rightsquigarrow (x, x+2) \quad E' \vdash (z, (x, x+1)) \vdash (f_n \ y \Rightarrow y5)(f_n \ z \Rightarrow 3) \rightsquigarrow 3 \quad (z, (x, x+1)) \vdash z \vdash [(f_n \ y \Rightarrow y5)(f_n \ z \Rightarrow 3)] \rightsquigarrow 4 \end{array}}{\emptyset \vdash f_n \ x \Rightarrow x+1 \rightsquigarrow (x, x+2)}$$

$\emptyset \vdash \text{let } z = (f_n \ x \Rightarrow x+1) \text{ in } z[(f_n \ y \Rightarrow y5)(f_n \ z \Rightarrow 3)] \rightsquigarrow 4$

$$\frac{\begin{array}{c} E' \vdash y \rightsquigarrow (z, 3) \quad E' \vdash s \rightsquigarrow E'(z, 5) \vdash 3 \rightsquigarrow 3 \\ E \vdash f_n \ y \Rightarrow y \ S \rightsquigarrow (y, y5) \quad E \vdash f_n \ z \Rightarrow 3 \rightsquigarrow (z, 3) \quad E' \vdash (y, (z, 3)) \vdash y5 \rightsquigarrow 12 \\ (x, 3) \vdash f_n \ z \Rightarrow z+x \rightsquigarrow (z, z+x) \quad E' \vdash (x, 3)(y, (z, z+x)) \vdash \text{let } x=7 \text{ in } y5 \rightsquigarrow 12 \\ \emptyset \vdash 3 \rightsquigarrow 3 \quad (x, 3) \vdash \text{let } y = (f_n \ z \Rightarrow z+x) \text{ in let } x=7 \text{ in } y5 \rightsquigarrow 12 \\ \emptyset \vdash \text{let } x=3 \text{ in let } y = (f_n \ z \Rightarrow z+x) \text{ in let } x=7 \text{ in } y5 \rightsquigarrow 12 \end{array}}{\emptyset \vdash \text{let } x=3 \text{ in let } y = (f_n \ z \Rightarrow z+x) \text{ in let } x=7 \text{ in } y5 \rightsquigarrow 12}$$

Semantica eager statica

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

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

es

$$\frac{\frac{\frac{(x,3)(z,5) \vdash z \rightsquigarrow 5 \quad (x,3)(z,7) \vdash x \rightsquigarrow 3}{(x,3)(z,5) \vdash z+x \rightsquigarrow 8} \quad E' \vdash y \rightsquigarrow (z, z+x, (x,3)) \quad E' \vdash 5 \rightsquigarrow 5}{(x,3) \vdash f_{\text{in}} z \Rightarrow z+x \rightsquigarrow (z, z+x, (x,3))} \quad E' \vdash (x,3)(y, (z, z+x), (x,3)) \vdash \text{let } x=7 \text{ in } y \rightsquigarrow 8}{\emptyset \vdash \text{let } x=3 \text{ in let } y=(f_{\text{in}} z \Rightarrow z+x) \text{ in let } x=7 \text{ in } y \rightsquigarrow 8}$$

non serve che ricevano
tanto è lo stesso ambiente

Espressione w

L'espressione w è una espressione non valutabile in alcuna semantica del linguaggio Fun

$$\frac{\emptyset \vdash f_n x \Rightarrow xx \rightsquigarrow (x,xx) \quad \emptyset \vdash f_n x \Rightarrow xx \rightsquigarrow (x,xx) \quad (x,(x,xx)) \vdash x \rightsquigarrow (x,xx) \quad (x,(x,xx)) \vdash xx \rightsquigarrow w}{\emptyset \vdash (f_n x \Rightarrow xx)(f_n x \Rightarrow xx) \rightsquigarrow w}$$

Non è valutabile poiché, come possiamo vedere, va in loop. Poco espressioni del tipo `let x=w in z` possono essere valutabili tramite una semantica lazy (in eager dovrei interagire con w e mi porterebbe in loop)
L'omega permette di introdurre la ricorsione, che rende il linguaggio Fun **Turing completo**

Semantica lazy statica

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

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

Curryificazione

La curryificazione consente nel trasformare una funzione che prende più argomenti in una sequenza di funzioni, ognuna delle quali prende un solo argomento

$$A \times B \rightarrow C \Rightarrow A \rightarrow (B \rightarrow C)$$

es 5+7

$$P \equiv f_n x \Rightarrow [f_n y \Rightarrow (x,y)]$$

↳ da implementare

Grazie a questo strumento è possibile semplificare il linguaggio Fun rimuovendo `let x=M in N`, infatti:
`let x=M in N` $\equiv (f_n x \Rightarrow N)M$

Numeri di Church

Nel λ -calcolo i numeri possono essere rappresentati come funzioni. In generale in numero n:

$$n \equiv \lambda f. \lambda x. f^n(x)$$

↳ applica f a x n volte

es SML

$$2 \equiv f_n x \Rightarrow f_n y \Rightarrow x(xy)$$

↳ DUE

adesso dobbiamo costruire una funzione eval per poterlo valutare

$$\text{eval} = (f_n z \Rightarrow z (f_n x \Rightarrow x+1) 0)$$

verifichiamo

$$\text{eval } 2 = (f_n z \Rightarrow z (f_n x \Rightarrow x+1) 0) \text{ DUE} =$$

$$= \text{DUE } (f_n x \Rightarrow x+1) 0 =$$

$$= (f_n x \Rightarrow f_n y \Rightarrow x(xy)) (f_n x \Rightarrow x+1) 0 = (f_n y \Rightarrow (f_n x \Rightarrow x+1)) ((f_n x \Rightarrow x+1) y) 0 =$$

$$= (f_n x \Rightarrow x+1) ((f_n x \Rightarrow x+1) 0) = (f_n x \Rightarrow x+1) 1 = 2$$

EAGER STATIC

vediamo ora la funzione per trovare il successore di un numero di Church
`succ = f_n z \Rightarrow f_n x \Rightarrow f_n y \Rightarrow x(z \times y)`

applichiamola

succ due =

$$\begin{aligned} &= (\text{fn } z \Rightarrow \text{fn } x \Rightarrow \text{fn } y \Rightarrow x(z \times y)) \text{ due} = \\ &= \text{fn } x \Rightarrow \text{fn } y \Rightarrow x(\text{due} \times y) = \\ &= \text{fn } x \Rightarrow \text{fn } y \Rightarrow x((\text{fn } x \Rightarrow \text{fn } y \Rightarrow x(xy)) \times y) = \\ &\quad \xrightarrow{(\text{fn } x \Rightarrow \text{fn } y \Rightarrow x(xy)) \times y = \text{fn } y \Rightarrow x(xy), y = x(xy)} \\ &= \text{fn } x \Rightarrow \text{fn } y \Rightarrow x(xy) = 3 \end{aligned}$$

altre operazioni

$$\text{false} \equiv \text{zero} \equiv \text{fn } x \Rightarrow \text{fn } y \Rightarrow y \quad \text{oppure fun zero } xy = y$$

$$\text{plus} \equiv \text{fn } v \Rightarrow \text{fn } v \Rightarrow (v \text{ succ } v) \quad \text{applico succ } v, v \text{ volte}$$

$$\text{allo stesso modo fun plus } uvxy = v \underset{v \text{ volte}}{x} (uxy) \underset{\text{succ}}{,}$$

$$\text{val times} = \text{fn } u \Rightarrow \text{fn } v \Rightarrow (u \underset{u \text{ volte}}{(f \underset{\text{addiziona}}{n } z \Rightarrow (\text{plus } z v)) \underset{\text{valore iniziale}}{\text{zero}}})$$

Linguaggio Imp

$K ::= \text{ol} \mid \text{t} \mid \text{f} \mid \text{false}$

$M, N ::= K \mid M + N \mid M \cdot N$

$p, q ::= \text{skip} \mid p; q \mid \text{if } M \text{ then } p \text{ else } q \mid x := M \mid \text{var } x = M \text{ in } p \mid \text{while } M \text{ do } p$

per proseguire è importante capire la differenza tra **call by reference** e **call by value**. La differenza è che quando passo un argomento in input a una funzione tramite call by value, viene creata una copia del valore passato (non viene modificata la variabile fuori dal contesto della funzione). Mentre nel call by reference, viene passato in input l'indirizzo di memoria della variabile (modificando la variabile dentro il contesto della funzione viene modificato anche fuori).

aggiungiamo adesso il concetto di locazione per prepararci al linguaggio All

$$E \in \text{Env} = \text{Var} \xrightarrow{\text{fin}} \text{Loc}$$

$$S \in \text{Store} = \text{Loc} \xrightarrow{\text{fin}} \text{Val}$$

Ci servono quindi due funzioni di valutazione della semantica in base a se sto valutando $M \in \text{Exp}$ o no:

$$1. \xrightarrow{M} \subseteq \text{Env} \times \text{Exp} \times \text{Store} \times \text{Val}$$

$$E \vdash M, S \xrightarrow{M} v$$

↳ è necessario lo store, altrimenti $E \vdash M$ darebbe una locazione (e non un valore)

$$2. \xrightarrow{M} \subseteq \text{Env}(x)$$

$E \vdash p, S \xrightarrow{M} S'$ → cambia quello che è memorizzato nello store

regole di inferenza

$$E \vdash K, S \xrightarrow{M} K$$

$$E \vdash x, S \xrightarrow{M} v \quad (\text{se } E(x) = l \text{ e } S(l) = v)$$

$$\begin{array}{c} E \vdash M, S \xrightarrow{M} v_1 \quad E \vdash N, S \xrightarrow{M} v_2 \\ E \vdash M + N, S \xrightarrow{M} v \end{array} \quad (\text{se } v_1 + v_2 = v)$$

$$\begin{array}{c} E \vdash M, S \xrightarrow{M} v_1 \quad E \vdash N, S \xrightarrow{M} v_2 \\ E \vdash M \cdot N, S \xrightarrow{M} \text{true} \end{array} \quad (\text{se } v_1 \cdot v_2 = v)$$

$$\begin{array}{c} E \vdash M, S \xrightarrow{M} v_1 \quad E \vdash N, S \xrightarrow{M} v_2 \\ E \vdash M \cdot N, S \xrightarrow{M} \text{false} \end{array} \quad (\text{se } v_1 \neq v_2)$$

$$E \vdash \text{skip}, S \xrightarrow{M} S$$

$$\begin{array}{c} E \vdash p, S \xrightarrow{M} S' \\ E \vdash q, S \xrightarrow{M} S'' \\ E \vdash p; q \xrightarrow{M} S'' \end{array}$$

$$\begin{array}{c} E \vdash M, S \xrightarrow{M} \text{true} \\ E \vdash \text{if } M \text{ then } p \text{ else } q, S \xrightarrow{M} S' \end{array}$$

$$\begin{array}{c} E \vdash M, S \xrightarrow{M} \text{true} \\ E \vdash p, S \xrightarrow{M} S' \\ E \vdash \text{if } M \text{ then } p \text{ else } q, S \xrightarrow{M} S' \end{array}$$

$$\begin{array}{c} E \vdash M, S \xrightarrow{M} \text{false} \\ E \vdash \text{while } M \text{ do } p, S \xrightarrow{M} S \end{array}$$

$$\begin{array}{c} E \vdash M, S \xrightarrow{M} \text{true} \\ E \vdash p, S \xrightarrow{M} S' \\ E \vdash \text{while } M \text{ do } p, S \xrightarrow{M} S' \end{array}$$

$$\begin{array}{c} E \vdash M, S \xrightarrow{M} v \\ E(x, l) \vdash p, S(l, v) \xrightarrow{M} S' \\ E \vdash \text{var } x = M \text{ in } p, S \xrightarrow{M} S' \end{array} \quad (\text{dove } l \text{ è "nuova"})$$

↑
definizione di una
nuova variabile

es. $(\text{var } x = M \text{ in } p); q$

cambia l'ambiente in cui sto valutando, ma solo localmente (fino a $;$); quando valuto q l'ambiente non cambia ma la locazione rimane nello store (non ci sta garbage collection)

$$\frac{E \vdash M, S \xrightarrow{H} v}{E \vdash x := M, S \xrightarrow{P} S(l, v)} \quad (\text{se } E(x) = l)$$

↑ riassegnamento a
variabile esistente

Linguaggio All

ampliamo Imp introducendo le **procedure** e gli **array**
 $\text{proc } y(x) \text{ is } p \text{ in } q \mid \text{call } y(M)$

parametro
di y ↑
cosa da
fare

il linguaggio All è statico o dinamico?

$\text{var } x := r \text{ in proc } y(z) \text{ is } w := x \text{ in var } x := z \text{ in call } y(z)$

- se dinamico $x = z$ (ultimo valore visto)
- se statico $x = r$ (primo valore visto)

specifiche di All: L-Exp e R-Exp

Una L-Exp è un'espressione che non si valuta a un semplice valore, ma a una locazione di memoria dove è memorizzato il valore (possono vivere alla sinistra di un simbolo di assegnamento, le R-Exp il contrario).

es

$x := s \rightarrow x$ è la L-Exp e s la R-Exp

viene quindi introdotta la L-Exp $V ::= x | x[M]$ e sono di conseguenza cambiate le regole di Imp

R-Exp

$M, N ::= K | V | M + N | M < N$

$p, q ::= \text{skip} | p; q | \text{if } M \text{ then } p \text{ else } q | V := M | \text{var } V = M \text{ in } p | \text{while } M \text{ do } p \text{ arr } x = [M_0, M_1, \dots, M_n] \text{ in } p | \text{proc } y(x) \text{ is } q \text{ in } p | \text{call } y(M)$

regole di inferenza

$$\frac{\forall v \in \text{Env} \times L\text{-exp} \times \text{Store} \times \text{Loc}}{M \in \text{Env} \times R\text{-exp} \times \text{Store} \times \text{Val}}$$

$$\frac{\forall v \in \text{Env} \times \text{All} \times \text{Store} \times \text{Store}}{M \in \text{Env} \times \text{All} \times \text{Store} \times \text{Loc}}$$

$$E \vdash x, S \rightsquigarrow l \quad (\text{se } E(x) = l)$$

$$\frac{\begin{array}{l} E \vdash M, S \rightsquigarrow m \\ E \vdash x[M], S \rightsquigarrow l_m \end{array}}{E \vdash x, S \rightsquigarrow l_m} \quad (\text{se } E(x) = \langle l_0, l_1, \dots, l_n \rangle \wedge m \leq n)$$

non serve $m > 0$ tanto non abbiamo i negativi

$$\frac{E \vdash V, S \rightsquigarrow l}{E \vdash V, S \rightsquigarrow v} \quad (\text{se } S(l) = v)$$

$$\frac{\begin{array}{l} E \vdash M, S \rightsquigarrow v \\ E \vdash V := M, S \rightsquigarrow S(l, v) \end{array}}{E \vdash V := M, S \rightsquigarrow S(l, v)} \quad \begin{array}{l} \text{se esiste } l \text{ viene} \\ \text{sostituito} \end{array}$$

$$\frac{\begin{array}{l} E \vdash M_i, S \rightsquigarrow v_i \\ E \vdash M, S \rightsquigarrow S(l_0, \dots, l_n) \end{array}}{E \vdash M, S \rightsquigarrow S(l_0, \dots, l_n, v_i)} \quad \begin{array}{l} \text{per tutti gli } M \\ \text{dove } l_0, \dots, l_n \end{array}$$

$$E \vdash \text{arr } x = [M_0, \dots, M_n] \text{ in } p, S \rightsquigarrow S'$$

$$\frac{E(y, (x, p, E)) \vdash q, S \rightsquigarrow S'}{E \vdash \text{proc } y(x) \text{ is } p \text{ in } q, S \rightsquigarrow S'}$$

call-by-value

$$\frac{\begin{array}{l} E \vdash M, S \rightsquigarrow v \\ E'(x, l) \vdash p, S(l, v) \rightsquigarrow S' \quad (\text{se } E(y) = (x, p, E')) \end{array}}{E \vdash \text{call } y(M), S \rightsquigarrow S'}$$

call-by-reference

$$\frac{\begin{array}{l} E \vdash V, S \rightsquigarrow l \\ E'(x, l) \vdash p, S \rightsquigarrow S' \quad (\text{se } E(y) = (x, p, E')) \end{array}}{E \vdash y(V), S \rightsquigarrow S'}$$

es) distinzione tra call-by reference e call-by value

$$\frac{E \vdash z, (l, 3) \rightsquigarrow_3 (z, l)(x, l') \vdash x := 2, (l, 3)(l', 3) \rightsquigarrow (l, 3)(l', 2)}{(z, l)(y, (x, x := 2, (z, l))) \vdash \text{call } y(z), (l, 3) \rightsquigarrow (l, 3)(l', 2)} E(y) = (x, x := 2, (z, l)) \quad \text{CBV}$$

$(z, l) \vdash \text{proc } y(x) \text{ is } x := 2 \text{ in call } y(z), (l, 3) \rightsquigarrow (l, 3)(l', 2)$

$$\frac{E \vdash z, (l, 3) \rightsquigarrow_3 (z, l)(x, l) \vdash x := 2, (l, 3) \rightsquigarrow (l, 2)}{(z, l)(y, (x, x := 2, (z, l))) \vdash \text{call } y(z), (l, 3) \rightsquigarrow (l, 2)} E(y) = (x, x := 2, (z, l)) \quad \text{CBR}$$

$(z, l) \vdash \text{proc } y(x) \text{ is } x := 2 \text{ in call } y(z), (l, 3) \rightsquigarrow (l, 2)$

call-by name (versione lazy di cbr)

$$E'(x, (V, E)) \vdash p, S \rightsquigarrow S'$$

$$E \vdash \text{call } y(V), S \rightsquigarrow S'$$

dove di conseguenza modificare Env

$$\text{Env: Var} \xrightarrow{\text{fin}} \text{Loc}^+ + (\text{Var} \times \text{All} \times \text{Env}) + (\text{All} \times \text{Env})$$

pezzo di sintassi
non validata (es. V)

dove inoltre aggiungere la seguente regola

$$E' \vdash V, S \rightsquigarrow b \quad (\text{se } E(x) = (V, E'))$$

$$E \vdash x, S \rightsquigarrow b$$

variable di riferimento
di una procedura

Cerchiamo adesso di capire come distinguere tra call-by name (lazy statico) e call-by reference (eager statico)

var $x=1$ in arr $z=[5, 6, 7]$ in proc $y(w)$ is $(x := 2; u := w)$ in call $y(z[x])$

$$L_{z[x]}, E'$$

$$U=6$$

$$U=7$$

statico: preserva gli ambienti
ma non gli store

Moltiplicazione egiziana

$138 \times$	<u>multiplico per 2</u>	<u>138</u>	<u>divido per 2</u>
<u>45 =</u>		90	69
690		180	47
<u>552 -</u>		360	17
6210		<u>720</u>	8
		1440	4
		<u>2880</u>	2
		5760	1
		6210	

Per decidere se un programma è corretto o meno devo stabilirlo non a tempo di esecuzione, ma guardando il programma

trasformiamolo in codice

```
public class Egypt {
    public static void main() {
        int a = 45;
        int b = 138;
        int x = a, y = b, res = 0;
        while (y >= 1) {
            if (y % 2 == 0) {
                x = x * 2;
                y = y / 2;
            } else {
                res = res + x;
                y = y - 1;
            }
        }
    }
}
```

abbiamo quindi che $a \cdot b = x \cdot y + res \wedge y \geq 0$

se entro nell'if so che l'invariante vale ancora (res non cambia, e $y/2$ sarà > 0 se lo era anche prima)

anche se entro nell'else rimane vero, infatti se $y=1$ (caso peggiore) rimane ≥ 0 e per quanto riguarda la moltiplicazione, è vero che moltiplico y una volta in meno ma allo stesso tempo incremento res di x

supponiamo ora che siamo usciti dal while, allora $y < 1$. Ma se $y < 1 \wedge y \geq 0 \Rightarrow y=0$, quindi $a \cdot b = x \cdot 0 + res = res$

Hoare logic

Vogliamo ora un linguaggio che ci permetta di dimostrare la correttezza dei programmi imperativi in modo più diretto.

$$M, N ::= K \mid M + N$$

$$A, B ::= \text{true} \mid \text{false} \mid A \rightarrow B \mid M, N \mid M = N$$

$$p, q ::= \text{skip} \mid p, q \mid x := M \mid \text{if } B \text{ then } p \text{ else } q \mid \text{while } B \text{ do } p$$

↑ non servirà
la dichiarazione

le altre regole
possono essere
derivate

$$\neg A \equiv A \rightarrow \text{false}$$

$$A \wedge B \equiv \neg(A \rightarrow \neg B)$$

$$A \vee B \equiv \neg A \rightarrow B$$

DEF Tripla di Hoare

Dato la logica di Hoare, siano A e B due espressioni booleane e sia p un programma. Se mi trovo in uno stato S che soddisfa A , allora l'esecuzione di p in S (cioè a partire dallo stato S) o va in loop, oppure termina in uno stato S' che soddisfa B .

Si scrive:

$$\{A\} p \{B\}$$

↑ precondizione ↑ postcondizione

Vediamo ora le **regole di inferenza**.

$$\{A\} p \{\text{true}\}$$
 in che modo la tripla interagisce con true

$$\{\text{false}\} p \{B\}$$
 in che modo la tripla interagisce con false
(non parte mai)

$$\frac{\text{strengthening}}{A \rightarrow B \quad \{B\} p \{C\}} \qquad \frac{\text{weakening}}{\{A\} p \{B\} \quad B \rightarrow C} \quad \{A\} p \{C\}$$

$$\frac{\{A\} p \{B_1\} \quad \{A\} p \{B_2\} \quad \dots \quad \{A\} p \{B_n\}}{\{A\} p \{B_1 \wedge B_2 \wedge \dots \wedge B_n\}}$$

soddisfa contemporaneamente
tutte le B

$$\frac{\{A_1\} p \{B\} \quad \{A_2\} p \{B\} \quad \dots \quad \{A_n\} p \{B\}}{\{A_1 \vee A_2 \vee \dots \vee A_n\} p \{B\}}$$

tutte le A_i vanno bene per avere
 B come postcondizione

$$\{A\} \text{skip} \{B\}$$

$$\frac{\{A \wedge B\} p \{C\} \quad \{A \wedge \neg B\} q \{C\}}{\{A\} \text{if } B \text{ then } p \text{ else } q \{C\}}$$

è debole (nonostante sia giusta)
infatti non tiene conto della
verità di B

$$\frac{\{A \wedge B\} p \{A\}}{\{A\} \text{while } B \text{ do } p \{A \wedge \neg B\}}$$

incorrettezza parziale se p termina la postcondizione è vera
(non garantisce che il ciclo finisce)

$$\frac{\{A\} p \{C\} \quad \{C\} q \{B\}}{\{A\} p; q \{B\}}$$

$$\{[M]x\} A \{x := M\}$$

↪ sostituzione di x con M

Si parla di **weakest precondition** quando la precondizione è minima affinché si possa avere la postcondizione.
Invece la **strongest precondition** è false (sto escludendo tutto)

es

$$\{A\} x := x + 1 \{x > 5\}$$

la weakest precondition è $A = x > 4$

Dimostriamo la correttezza della divisione euclidea

```

b' rest:  $\frac{x}{y}$ 
b := x;
a := 0;
risultato
while (b >= y) do
  (b := b - y;
  a := a + 1;
)
  
```

ovvero vogliamo dimostrare che i valori a e b rispettano
 $A: x = ay + b \wedge b \geq 0 \wedge b \leq y$ (dimostriamo che sono un invarianti)
usando come precondizione $\{x \geq 0\}$ dato che se fosse $x < 0$ non entrò mai nel while

Procediamo ora per passi e verifichiamo il programma

non lo dimostriamo perché
che in Hoare è dimostrabile
(sintassi astratta)

$$(x \geq 0) \rightarrow (x = x \wedge x \geq 0) \quad \{x = x \wedge x \geq 0\} \{b := x \wedge b \geq 0\}$$

$$\{x \geq 0\} \{b := x \wedge b \geq 0\}$$

[rafforzamento]

$$(b = x \wedge b \geq 0) \rightarrow (b = x \wedge b \geq 0) \quad \{b = x \wedge b \geq 0\} \{a := 0 \wedge ay + b = x \wedge b \geq 0\}$$

$$\{b = x \wedge b \geq 0\} \{a := 0 \wedge ay + b = x \wedge b \geq 0\}$$

B l'implicazione $x \geq 0$

$$\{x \geq 0\} \{b := x \wedge B\} \quad \{B\} \{a := 0 \wedge A\}$$

$$\{x \geq 0\} \{b := x; a := 0 \wedge A\}$$

Procediamo entrando nel while

$$\{A \wedge b \geq y \wedge b := b - y; a := a + 1 \wedge A\}$$

$$\{A\} \text{ while } (b \geq y) \text{ do } (b := b - y; a := a + 1) \{A \wedge b \geq y\}$$

ora se dimostro che A è vero almeno alla fine del primo ciclo ho finito in quanto A è il nostro supposto invarianti (quindi rimarrà sempre vero)

separo ulteriormente

$$(C) \rightarrow (x = (a+1)y + b - y \wedge b - y \geq 0) \quad \{x = (a+1)y + b - y \wedge b - y \geq 0\} \{a := a + 1 \wedge A\}$$

$$\{C\} \{a := a + 1 \wedge A\}$$

$$(A \wedge b \geq y) \rightarrow (x = a y + b - y \wedge b - y \geq 0) \quad \{x = a y + b - y \wedge b - y \geq 0\} \{b := b - y \wedge A \wedge b \geq 0\}$$

$$\{A \wedge b \geq y\} \{b := b - y \wedge A \wedge b \geq 0\}$$

$$\left. \begin{array}{l} x = (a+1)y + b - y \wedge b - y \geq 0 \\ \Rightarrow x = ay + y + b - y \wedge b \geq y \\ \Rightarrow x = ay + b \wedge b \geq y \\ \Rightarrow x = ay + b \wedge b \geq 0 \end{array} \right\} \Rightarrow A \text{ è valido alla fine del while}$$

↓
è un INVARIANTE

DEF Punto fisso

data una endofunzione $f: X \rightarrow X$ e $x \in X$, definiamo x come punto fisso di f se $f(x) = x$

OSS non è vero che ogni endofunzione $\mathbb{N} \rightarrow \mathbb{N}$ ha un punto fisso, ma per ogni funzione Dominio \rightarrow Dominio (es. int) esiste un punto fisso

DEF Combinatore di punto fisso

All'interno del lambda calcolo, definiamo combinatore di punto fisso come

$$Y \equiv f_n G \Rightarrow [(f_n x \Rightarrow G(x x))(f_n x \Rightarrow G(x x))]$$

Y è la funzione che restituisce il punto fisso di una data funzione

TEOREMA

Data una funzione F , Yh applica h ricorsivamente

dim

$$YF \equiv [f_n G \Rightarrow (f_n x \Rightarrow G(x x))(f_n x \Rightarrow G(x x))] F \xrightarrow{\beta} (f_n x \Rightarrow F(x x))(f_n x \Rightarrow F(x x)) \xrightarrow{\beta} F [(f_n x \Rightarrow F(x x))(f_n x \Rightarrow F(x x))] = F(YF)$$

OSS usiamo la logica equazionale per i programmi funzionali, in particolare qui si usa la β -regola del lambda calcolo
 β -regola $(f_n x \Rightarrow M) N = [N/x] M$

COROLLARIO dando in input a una funzione F il suo punto fisso tramite il combinatore mi viene restituito il punto fisso stesso
 $F(YF) = YF$

es fattoriale

$$Ff \equiv \text{if } (n=0) \text{ then } z \text{ else } n \cdot f(n-1)$$

se f è il fattoriale, allora F è il fattoriale e dunque la funzione f (fattoriale) è il punto fisso di F

infatti, visto che $Y:(A \rightarrow A) \rightarrow A$ dove $(A \rightarrow A)$ è l'insieme delle funzioni che prendono una funzione e restituiscono una funzione nello stesso dominio, quindi A è una funzione e quanto detto ha senso

Espressioni booleane di Church

Oltre ai numeri è possibile anche descrivere la logica booleana di Hoare

true fn $x \Rightarrow x$

false fn $x \Rightarrow y$

not fn $z \Rightarrow (fn \ xy \Rightarrow z \ y \ x)$

bool
di Church

restituzione gli argomenti
invertiti, true se era false e
viceversa

eval bool fn $z \Rightarrow z$ true false

Grazie ai booleani adesso mi è possibile costruire una funzione ricorsiva per determinare se un numero è pari o meno

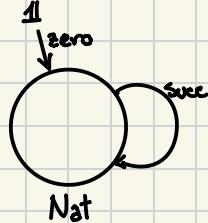
fun is-even 0=true | is-even(succ n)=not(is-even n) funzione per clausole in SML
 ↳ clausola 1 ↳ clausola 2

Verifica di specifiche e quozientali

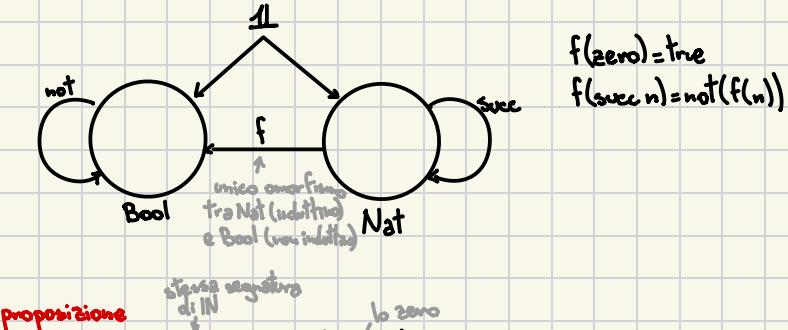
Definiamo un'algebra induttiva Nat in SML attraverso il costrutto datatype

new type
datatype Nat = zero of unit | succ of Nat

rimane valido nonostante
seci non sia stato definito
infatti se eseguiamo
succ(zero()) sappiamo
solamente che è Nat



Per quanto visto a inizio corso, è possibile definire una funzione onomorfa tra un'algebra induttiva e una non induttiva con stessa segnatura



Dati un insieme A , un elemento $a \in A$ ed una funzione $h: A \rightarrow A$, esiste una e una sola $f: \text{IN} \rightarrow A$ t.c. $f(0) = a$ e $f(\text{succ}(n)) = h(f(n))$

Indichiamo con ρ la funzione che, dati $M(a)$ e $N(h)$ restituisce l'omomorfismo f . Aggiungiamo ρ al seguente linguaggio

$M, N ::= 0 \mid x \mid \text{fn } x \Rightarrow M \mid MN \mid \text{succ}(M) \mid \rho(M, N)$

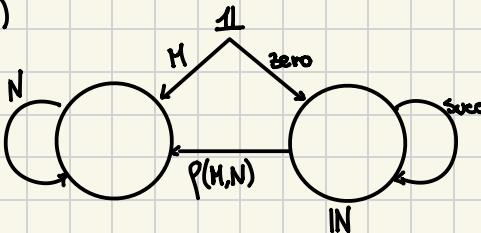
dove:

$\rho(M, N) 0 = M$

assiomi specifici

$\rho(M, N)(\text{succ } L) = N[\rho(M, N)L]$

gli assiomi generali sono: riflessività
simmetria e transitività



Caveat β -regola VARIABLE CATCHING

La β -regola è call-by name, non posso "borrivamente" sostituire le variabili, ma devo fare attenzione ad eventuali variabili omoniime

(es)

$[\text{fn } x \Rightarrow (\text{fn } y \Rightarrow x + y)]y$

$\text{fn } y \Rightarrow y + y$

sostituendo "borrivamente" invece ho

$\text{fn } y \Rightarrow y + y$

$[\text{fn } y \Rightarrow \text{fn } y \Rightarrow y]z \equiv [\text{fn } y \Rightarrow ?]z \equiv ?$

$[\text{fn } y \Rightarrow \text{fn } y \Rightarrow y]z \equiv ?z \equiv ?$

esempio sbagliato

$$\frac{P(0) \quad P(x) \rightarrow P(\text{succ } x)}{\forall x \ P(x)}$$

$$(f_n \ x \Rightarrow M) N \stackrel{?}{=} [N/x] M$$

$$\{[M/x] A\} x := M \{A\}$$

α -equivalenza CAMBIARE NOME VARIABILE

$f_n \ x \Rightarrow M = f_n \ y \Rightarrow [y/x] M$ due termini sono uguali
se presi due termini sostituisco le variabili nell'espressione

Esercizio

Verifichiamo la correttezza della moltiplicazione per 2

$$\begin{aligned} \text{datatype Nat} &= \text{zero} \mid \text{succ of Nat} \\ \text{twice zero} &= \text{zero} \mid \text{twice}(\text{succ } n) = \text{succ}(\text{succ}(\text{twice}(n))) \\ \text{twice} &\equiv \text{rec}(\text{zero}, f_n \ xy \Rightarrow \text{succ}(\text{succ}(x))) \end{aligned}$$

$$\begin{cases} 5 \cdot 2 = 10 \\ 6 \cdot 2 = 12 \\ 12 - 10 = 2 \end{cases}$$

verifichiamo ora la correttezza per farlo, abbiamo necessità di $f_n \text{ plus zero } n = n \mid \text{plus}(\text{succ}(m), n) = \text{succ}(\text{plus } m \ n)$

$$5+4 = (4+4)+1$$

TEOREMA

$$\text{twice} \equiv \text{plus } n \ n$$

dim

$$\text{base: twice } 0 = \text{plus } 0 \ 0$$

$$\begin{aligned} \text{per costruttivo: twice}(\text{succ}(n)) &= \text{plus}(\text{succ}(n), \text{succ}(n)) = \\ &= \text{succ}(\text{succ}(\text{twice } n)) = \\ &= \text{succ}(\text{succ}(\text{plus } n \ n)) = \\ &= \text{succ}(\text{plus}(\text{succ}(n), n)) \underset{\text{comutatività}}{=} \\ &= \text{succ}(\text{plus}(n, \text{succ}(n))) = \text{plus}(\text{succ}(n), \text{succ}(n)) \end{aligned}$$

TEOREMA

$$\text{plus}(n \ m) = \text{plus}(m) \ n$$

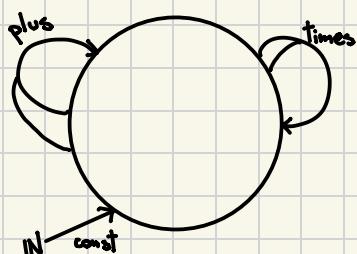
dim

$$\text{base: plus}(0 \ m) = \text{plus}(m) \ 0$$

$$\begin{aligned} \text{per costruttivo: hp ind plus}(\text{succ}(n) \ m) &= \text{plus}(m \ \text{twice}(n)) \ \alpha \\ \text{plus}(\text{succ}(n), \text{succ}(m)) &= \\ &= \text{succ}(\text{plus}(n \ \text{succ}(m))) = \text{succ}(\text{plus}(\text{succ}(m), n)) = \\ &= \text{succ}(\text{succ}(\text{plus}(m \ n))) = \text{succ}(\text{succ}(\text{plus}(m \ n))) = \\ &= \text{succ}(\text{plus}(\text{succ}(n) \ m)) = \text{succ}(\text{plus}(m \ \text{succ}(n))) = \\ &= \text{plus}(\text{succ}(m) \ \text{succ}(n)) \end{aligned}$$

Datatype in SKL

datatype Exp = const of int | plus of (Exp * Exp) | times of (Exp * Exp);



es

$$8+2 \Rightarrow \text{plus}(\text{const}(8), \text{const}(2))$$

fun eval(const n) = n | eval(plus(m, n)) = (eval m) + (eval n) | eval(times(m, n)) = eval(m) * eval(n);

Sistema dei tipi

Il sistema dei tipi è un sistema logico composto da un insieme di regole che assegna una proprietà detta **tipo** ad ogni termine di un linguaggio.

FATTO

non è possibile che esista una A che ha la stessa cardinalità di $A \rightarrow B$ per quanto possa essere piccolo B . Infatti: $|A \rightarrow B|$ è la cardinalità di $\wp(A)$ che non ha la proprietà del numerabile ($\gg |A|$)

Lambda calcolo tipato semplice F₁

$$M, N ::= K | x | MN | f_n x : A \Rightarrow M$$

\hookrightarrow tipo di x
(non specifica tipo
di output)

$$A, B ::= K | A \rightarrow B$$

↑
es. int
bool

es

$$\frac{\text{bool} \rightarrow \text{bool}}{\{[(f_n x : \text{int} \Rightarrow (f_n y : \text{bool} \Rightarrow y))] s\} \text{ true}}$$

$\frac{}{\text{int} \rightarrow (\text{bool} \rightarrow \text{bool})}$

$$\frac{\text{int} \rightarrow \text{int}}{[f_n x : \text{int} \Rightarrow [(f_n z : \text{int} \Rightarrow z) x]]}$$

$\frac{}{\text{int} \rightarrow \text{int}}$

Adesso però formalizziamo quello che abbiamo fatto ad occhio.

DEF Judgements

Usiamo Γ per indicare il **contesto dei tipi**, ovvero una lista ordinata $x_1 : A_1, x_2 : A_2, \dots, x_n : A_n$ di variabili x_i distinte, ciascuna associata ad un tipo A_i :

Il sistema F₁ permette di dedurne **asserzioni di tipo**, ovvero clausole della forma $\Gamma \vdash H : A$

regole di inferenza

$\Gamma : K : K \rightarrow$ per ogni K assumo di sapere il tipo

$$\frac{\Gamma_1, x : A, \Gamma_2 \vdash x : A}{\Gamma \vdash x : A}$$

$$\frac{\Gamma \vdash M : A \rightarrow B \quad \Gamma \vdash N : A}{\Gamma \vdash MN : B}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash (f_n x : A \Rightarrow M) : A \rightarrow B}$$

es

$$\frac{\Gamma \vdash x : \text{int} \rightarrow \text{bool} \quad \Gamma \vdash s : \text{int}}{\emptyset \vdash (f_n x : \text{int} \rightarrow \text{bool} \Rightarrow x s) : (\text{int} \rightarrow \text{bool}) \rightarrow \text{bool}}$$

$\frac{x : \text{int} \rightarrow \text{bool} \vdash x s : \text{bool} \quad y : \text{int} \vdash \text{true} : \text{bool}}{\emptyset \vdash (f_n y : \text{int} \Rightarrow \text{true}) : \text{int} \rightarrow \text{bool}}$

$$\emptyset \vdash [(f_n x : \text{int} \rightarrow \text{bool} \Rightarrow x s)(f_n y : \text{int} \Rightarrow \text{true})] : \text{bool}$$

Il problema in F₁ si nota quando proviamo a valutare qualcosa del tipo $f_n x : A \Rightarrow x x : A \rightarrow B$ infatti:

$$\frac{\begin{array}{c} x : A \vdash x : A \rightarrow B \quad x : A \vdash x : A \\ x : A \vdash x x : B \end{array}}{\emptyset \vdash f_n x : A \Rightarrow x x : A \rightarrow B}$$

ma ciò è possibile se $A = A \rightarrow B$. Ciò però avviene solo considerando gli Stream ($A \rightarrow (A \rightarrow (A \rightarrow (\dots)))$ in questo caso infatti $T = T \rightarrow A$ è semanticamente corretto, ma sintatticamente non ha senso)

Visto che non è tipabile in F_1 , non posso neanche il combinatore di punto fisso (inoltre qualsiasi cosa in F_1 termina)

Polimorfismo parametrico

Il lambda calcolo tipato semplice viene considerato un sistema dei tipi di primo ordine, in quanto tale non permette di descrivere i tipi generici.

Invece nei sistemi dei tipi di secondo ordine, le variabili possono assumere tipi polimorfi, ossia quantificati universalmente

es

$$(fn\ x \Rightarrow (fn\ y \Rightarrow \text{not}\ y)) \quad \forall X. X \rightarrow (\text{bool} \rightarrow \text{bool})$$

Lambda calcolo polimorfo F_2

$$M, N ::= K \mid I \mid fn\ x:A \Rightarrow M \mid MN \mid \Delta X. M \mid MB$$

$$A, B ::= K \mid A \rightarrow B \mid X \mid \Delta X. A$$

↳ introducete le variabili nei tipi

operatori di astrazione legano rispettivamente M e A

regole di inferenza

ottene alle regole di F_1

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash \Delta X. M : \forall X. A} \quad \text{(se } X \notin \text{free}(\Gamma) \text{)} \quad \text{[generalizzazione dei tipi]}$$

es.

$$\begin{array}{l} \vdash M : \forall X. X \rightarrow \text{bool} \\ \vdash M[\text{int}] : \text{int} \rightarrow \text{bool} \end{array}$$

l'unico modo in cui M è del tipo $\forall X. A$ è quando M è della forma Δ a causa della regola di generalizzazione

es lambda astrazione

$\forall X. X \rightarrow \text{bool}$ fa astrazione rispetto al tipo della variabile X
mentre $fn\ x:A \Rightarrow M$ lega la variabile x e richiede che il suo tipo A sia noto nel contesto

per identità

dunque se ad esempio ho $(fn\ x:X \Rightarrow x) s$ per com'è non può funzionare infatti: ho un type mismatch ($s:\text{int}$, $x:X$) e dunque devo utilizzare l'operatore di astrazione, diventando:

$$\{[\Delta X. fn\ x:X \Rightarrow x] int\} s$$

facciamo l'albero di inferenza

$$\begin{array}{l} (x:X) \vdash x:X \\ \hline \emptyset \vdash fn\ x:X \Rightarrow x : X \rightarrow X \\ \hline \emptyset \vdash \Delta X. fn\ x:X \Rightarrow x : \forall X. X \rightarrow X \\ \hline \emptyset \vdash (\Delta X. fn\ x:X \Rightarrow x)[\text{int}] : \text{int} \rightarrow \text{int} \quad \emptyset \vdash s : \text{int} \\ \hline \emptyset \vdash [(\Delta X. fn\ x:X \Rightarrow x)[\text{int}]] s : \text{int} \end{array}$$

polimorfico con esplicitazione dei tipi: $\Delta X (fn\ x:X \Rightarrow x)$ → in questo caso se gli passo un tipo lavoro su quella funzione con il tipo di x

in questo caso x è polimorfo infatti prima gli passo una funzione del tipo $A \rightarrow A$ e dopo un valore booleano (non avendo constraint sul tipo di x lo posso fare)

consideriamo invece adesso

$$\text{let } x = (fn\ y \Rightarrow y) \text{ in } [x (fn\ z \Rightarrow z)] (\text{true})$$

Vediamo ora perché la regola della generalizzazione ha la condizione che $X \notin \text{free}(\Gamma)$

es) $\text{free}(\Gamma') = X$

$(x:X) \vdash x:X$

$(x:X) \vdash \Lambda X.x:\forall X.X$

$(x:X) \vdash (\Lambda X.x)[\text{int}]:\text{int}$

$\emptyset \vdash f_n x:X \Rightarrow (\Lambda X.x)[\text{int}]:X \rightarrow \text{int}$

$\emptyset \vdash \Lambda X.f_n x:X \Rightarrow (\Lambda X.x)[\text{int}]:\forall X.X \rightarrow \text{int}$

sto violando la condizione

$X \notin \text{free}(\Gamma')$ quindi posso
assegnare a x : tipo int

(assurdo, x è di tipo X)

sto sviluppando
in questa direzione

per esempio tipo specifico

ad esempio $\text{free}(z:\forall Z.Z \rightarrow X) = \{X\}$
mentre $\text{free}(X \rightarrow Y) = \{X, Y\}$ e
 $\text{free}(x:X) = X$

lo può essere di un tipo qualsiasi

Sistema dei tipi di ML

$M, N ::= K | x | \text{fn } x \Rightarrow M | MN | \text{let } x = M \text{ in } N$

$\text{let } x = M \text{ in } N \equiv (\text{fn } x \Rightarrow N)M ?$

operazionalmente sono uguali, ma sono diversi nel sistema dei tipi infatti:

$\text{let } x = (\text{fn } y \Rightarrow y) \text{ in } xx \not\equiv (\text{fn } x \Rightarrow xx)(\text{fn } y \Rightarrow y),$

$\hookrightarrow \text{fn } y \Rightarrow y : \forall A. A \rightarrow A$ e il let in
ML generalizza, quindi in xx
il primo $x : (A \rightarrow A) \rightarrow (A \rightarrow A)$
mentre il secondo $x : A \rightarrow A$

\hookrightarrow in questo caso invece ci sta fn e quindi x non
è polimorfo quindi per la regola dell'applicazione
si avrebbe che in xx la seconda $x : T$ mentre la
prima $x : T \rightarrow U$ quindi implica che $T = T \rightarrow U$
paradosso di Curry

sintassi dei tipi di ML (Fun_τ)

$\tau ::= K | X | \tau_1 \rightarrow \tau_2$ [tipi primitivi]
 $\sigma ::= \tau | \forall X. \sigma$ [schema dei tipi]

τ non ha generalizzazioni

in questo caso i tipi non compaiono nella sintassi dei tipi

es

$\forall X[(\text{int} \rightarrow X) \rightarrow Y]$	valido
$(\forall X. X) \rightarrow (\forall Y. Y)$	valido in F_2 ma non qui infatti adesso non posso fare una funzione che prende in input un oggetto polimorfo (non posso tipare $\text{fn } x \Rightarrow xx$) questo invece è valido ma non abitabile (non esiste termine in ML con questo tipo)
$\forall X. \forall Y. X \rightarrow Y$	

$\text{fn } x : \forall X. X \rightarrow x(\text{int} \rightarrow \text{int})(x : \text{int})$ dal punto di vista di F_2 è perfettamente tipabile, però in ML non è valido
dato che non posso prendere in input un tipo polimorfo. Per questo let e
fn non sono equivalenti

regole di inferenza

$$\frac{\Gamma \vdash M : \tau' \rightarrow \tau \quad \Gamma \vdash N : \tau'}{\Gamma \vdash MN : \tau}$$

$$\frac{\Gamma, x : \tau \vdash M : \tau'}{\Gamma \vdash \text{fn } x \Rightarrow M : \tau \rightarrow \tau'} \text{ no tipo polimorfo}$$

$$\frac{\Gamma \vdash M : \sigma \quad \Gamma, x : \sigma \vdash N : \sigma'}{\Gamma \vdash \text{let } x = M \text{ in } N : \sigma'} \text{ polimorfo}$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M : \forall X. \sigma} (x \notin \text{free}(\Gamma))$$

$$\frac{\Gamma \vdash M : \sigma \quad (\text{se } \sigma' \subset \sigma)}{\Gamma \vdash M : \sigma'} \text{ se } \sigma' \text{ è un'istanza di } \sigma \text{ (sottotipo)}$$