

Appunti Semantics

Diego Oniarti

Anno 2024-2025

Contents

1	Lambda Calculus	2
1.1	Sintassi	2
2	SOS - Structural Operational Semantics	3
3	SOS - Call By Name	5
4	Big Step	5
4.1	Equivalenza con SS	5
5	Contextual Operation Semantics	6
5.1	COS, SS, CBV	6
6	Teorema di equivalenza SOS e COS	6
6.1	Prova per induzione del lemma 1	6
6.2	Prova per definizione del lemma 2	7
7	Simply Typed Lambda Calculus	8
8	Expanding The STLC	9
8.1	Aggiungere tuple	9
8.2	Aggiungere inums	10
8.3	Booleani	11
9	If Then Else	13
10	Properties of STLC	13
10.1	Type soundness	13
10.1.1	Progress	13
10.1.2	Preservation	13
10.2	Normalization	14
10.3	proofs	14
10.3.1	Proof of Progress	14
10.3.2	Proof of Preservation	14

10.3.3 Proof of Normalization	15
11 Logical Relationships (and semantic typing)	16
12 Proof of Normalization	16
13 lemma: vals in terms	17
14 Compatibility lemmas	17
14.1 Application	17
15 Introduction and Destruction	17
15.1 logica	17
16 System F	18
16.1 Existential Types	19

1 Lambda Calculus

Modello formale per il calcolo funzionale.

Il "While Language" (?) è più o meno la stessa cosa ma per la programmazione procedurale, che non faremo.

1.1 Sintassi

Sintassi per l'Untyped Lambda Calculus (ULC):

$$\begin{aligned}
 t := & n \in \mathbb{N} \\
 & | t \oplus t \\
 & | \lambda x. t \\
 & | x \in X \\
 & | t \ t
 \end{aligned}$$

dove:

- t è una metavariable
- $:=$ è "RNF" (?)
- \oplus è $+$, $-$, e \times
- λ indica una funzione, in questo caso con parametro x e body t .
- Tutto è associativo a sinistra

Questo vuol dire che un termine nel nostro linguaggio è un numero naturale o una somma di termini.

nb. Possiamo fare delle semplificazioni come usare n per rappresentare i numeri reali invece che preoccuparci della rappresentazione binaria.

example: $(\lambda x.x + 1) 3$ Questo rappresenta una funzione "successivo" e invoca la funzione sul numero 3.

2 SOS - Structural Operational Semantics

$$\begin{array}{l}
 t ::= n \\
 | t \oplus t \\
 | \lambda x.t \\
 | x \in X \\
 | t
 \end{array}$$

$$\begin{array}{l}
 \text{program state} \\
 \underbrace{\Omega} ::= t \\
 | fail
 \end{array}$$

We can divide terms in **redexes** and **values**.

Redexes

- $n \oplus n$
- $(\lambda x.t) v$

Values

$$\begin{array}{l}
 v ::= n \\
 | \lambda x.t
 \end{array}$$

Redexes change the state of the program according to some rules:

rules

$$\begin{array}{ll}
 \frac{[n \oplus n'] = n''}{n \oplus n' \rightarrow n''} & \text{sos-bop} \\
 \frac{(\lambda x.t)v \rightarrow t[\frac{v}{x}]}{t \rightarrow t''} & \text{sos-beta} \\
 \frac{t \rightarrow t''}{t \oplus t' \rightarrow t'' \oplus t'} & \text{sos-bop-1}
 \end{array}$$

$$\begin{array}{c}
\frac{t \rightarrow t'}{n \oplus t \rightarrow n \oplus t'} \quad \text{sos-bop-2} \\
\frac{t \rightarrow t''}{t \ t' \rightarrow t'' \ t'} \quad \text{sos-app-1} \\
\frac{t' \rightarrow t''}{(\lambda x.t)t' \rightarrow (\lambda x.t) \ t''} \quad \text{sos-app-2}
\end{array}$$

substitution

$$\begin{aligned}
n[v/x] &= n \\
x[v/x] &= v \\
y[v/x] &= y
\end{aligned}$$

$$\begin{aligned}
(t \oplus t')[v/x] &= t[v/x] \oplus t'[v/x] \\
(t \ t')[v/x] &= t[v/x] \ t'[v/x] \\
(\lambda y.t)[v/x] &= \lambda y. t[v/x]
\end{aligned}$$

Ogni regola modifica lo stato del programma, quindi possiamo dire abbiano la forma $\Omega \rightarrow \Omega$. Un programma corretto risolve a un *valore* dopo una serie di "steps".

Errori Programmi come "5 4" o " $0 + (\lambda x.x)$ " sono ben formati dal punto di vista della grammatica indicata. Portano però a delle redex a cui non si può applicare alcuna regola.

Aggiungiamo quindi uno stato "*fail*" a Ω e delle regole per propagare questo fail.

Fails

$$\begin{array}{c}
\frac{}{(\lambda x.t) \oplus t \rightarrow \text{fail}} \text{ sos-f-L} \\
\frac{}{n \ t \rightarrow \text{fail}} \text{ sos-f-n} \\
\frac{}{n \oplus \lambda x.t \rightarrow \text{fail}} \text{ sos-f-L2} \\
\\
\frac{t \rightarrow t'' \ t'' \rightarrow \text{fail}}{t \oplus t' \rightarrow \text{fail}} \text{ sos-bop-f1} \\
\frac{t \rightarrow t' \ t' \rightarrow \text{fail}}{n \oplus t \rightarrow \text{fail}} \text{ sos-bop-f2} \\
\frac{t \rightarrow t'' \ t'' \rightarrow \text{fail}}{t \ t' \rightarrow \text{fail}} \text{ sos-app-f1}
\end{array}$$

$$\frac{t' \rightarrow t'' \quad t'' \rightarrow fail}{(\lambda x.t) t' \rightarrow fail} \text{ sos-app-f2}$$

3 SOS - Call By Name

We don't apply a function to values but to symbols. The symbols are then lazily evaluated when they're used.

$$\Omega \xrightarrow{N} \Omega$$

Let's see which rules change under these new assumption:

$$\begin{array}{ll} \frac{}{n \oplus n' \xrightarrow{N} n''} & \text{sos-bop N} \\ \frac{}{(\lambda x.t) t' \xrightarrow{N} t[\frac{t'}{x}]} & \text{sos-beta N} \\ \text{untouched} & \text{sos-app1N} \\ \text{untouched} & \text{sos-bop1N} \\ \text{untouched} & \text{sos-bop2N} \end{array}$$

4 Big Step

Una semantica *big step* ha un judgement del tipo:

$$t \Downarrow v$$

Questo vuol dire che le inference rules non fanno più pattern matching su $\Omega \rightarrow \Omega$ ma su $t \Downarrow v$ (il termine t riduce a un valore v).

rules:

$$\begin{array}{ll} \frac{}{v \Downarrow v} & \text{val} \\ \frac{t \Downarrow n \quad t' \Downarrow n' \quad n \oplus n' = n''}{t \oplus t' \Downarrow n''} & \text{bs-bop} \\ \frac{t \Downarrow \lambda x.t'' \quad t' \Downarrow v \quad t''[v/x] \Downarrow v'}{t t' \Downarrow v'} & \text{bs-app} \end{array}$$

4.1 Equivalenza con SS

Big Step e Small Step sono equivalenti. Questo vuol dire che ogni termine che riduce a un valore in big step, converge allo stesso valore in small step. Questo è utile per alcune dimostrazioni, in quanto possiamo usare la struttura ad albero di BS nelle dimostrazioni per SS.

5 Contextual Operation Semantics

5.1 COS, SS, CBV

Chiamiamo E l'*evaluation context*, così definito.

$$\begin{aligned}
 E ::= & [] \\
 & | E \ t \\
 & | (\lambda x. t) E \\
 & | E \oplus t \\
 & | n \oplus E
 \end{aligned}$$

Abbiamo poi 2 judgements

$$\begin{array}{ll}
 \Omega \rightsquigarrow \Omega & \text{main reduction} \\
 \Omega \rightsquigarrow^P \Omega & \text{primitive reduction}
 \end{array}$$

$$\begin{array}{c}
 \frac{t \rightsquigarrow^P t'}{E[t] \rightsquigarrow E[t']} \text{ ctx} \\
 \frac{}{n \oplus n' \rightsquigarrow^P n''} \text{ c-bop} \\
 \frac{}{(\lambda x. t) v \rightsquigarrow^P t[v/x]} \text{ c-beta}
 \end{array}$$

esercizio. $((\lambda x. \lambda y. \lambda z. z \ x - y \ x) 5)(\lambda v. v)(\lambda w. 2 * w)$

wow. SOS e COS risolvono un'espressione con lo stesso numero di passaggi

6 Teorema di equivalenza SOS e COS

$$\forall t, t'. t \rightarrow t' \iff t \rightsquigarrow t'$$

Per ogni coppia di termini t e t' , t fa uno step SOS a t' se e solo se t fa anche uno step COS a t' . Per dimostrare l'*iff* dimostriamo prima il \implies e poi l' \impliedby .

lem.1 $\forall t, t'. t \rightarrow t' \implies t \rightsquigarrow t'$

lem.2 $\forall t, t'. t \rightarrow t' \impliedby t \rightsquigarrow t'$

6.1 Prova per induzione del lemma 1

Usiamo i termini come struttura induttiva. Se vediamo i termini come il loro Abstract Syntax Tree, possiamo partire da termini la cui altezza è zero e costruirne altri più complessi per induzione.

L'altra struttura induttiva che possiamo usare è la derivazione SOS. Anche essa è un albero, quindi lo stesso ragionamento vale.

Iniziamo quindi con i casi base. In questo caso abbiamo solo *bop* e *beta*.

- BOP

$$\begin{aligned}
 & t = n \oplus n' \quad t' = n'' \\
 \text{TS: } & n \oplus n' \rightsquigarrow n'' \\
 & \text{by ctx with } E = [] \\
 \text{TS: } & n \oplus n' \rightsquigarrow^P n'' \\
 & \text{by c-bop}
 \end{aligned}$$

- BETA

$$\begin{aligned}
 & t = (\lambda x. t'')v \quad t' = t''[v/x] \\
 \text{TS: } & (\lambda x. t'')v \rightsquigarrow t''[v/x] \\
 & \text{by ctx with } E = [] \\
 \text{TS: } & (\lambda x. t'')v \rightsquigarrow^P t''[v/x] \\
 & \text{by c-beta}
 \end{aligned}$$

Dimostriamo ora il passo induttivo per la prova del lemma 1:

In questo caso avremmo 4 casi induttivi da dimostrare (bop1, bop2, app1, app2) ma ne facciamo uno (app1) solo per brevità.

$$\text{TH: } \forall t_h, t'_h \text{ if } t_h \rightarrow t'_h \text{ then } t_h \rightsquigarrow t'_h$$

- app1: $t = t_1 \ t_2 \quad t' = t'_1 \ t_2$

$$\begin{aligned}
 \text{TH: } & t_1 \ t_2 \rightsquigarrow t'_1 \ t_2 \\
 \text{HP1: } & t_1 \ t_2 \rightarrow t'_1 \ t_2 \\
 \text{HP2: } & t_1 \rightarrow t'_1 \\
 & \text{by IH with HP2 wh } t_1 \rightsquigarrow t'_1 \quad \text{HT1} \\
 & \text{by inversion on HT1 wh } \begin{cases} t_1 \equiv E[t_0] & \text{HE1} \\ t'_1 \equiv E[t'_0] & \text{HE1'} \\ t_0 \rightsquigarrow^P t'_0 & \text{HPR} \end{cases} \\
 & \text{by HE1, HE1' TS } E[t_0] \ t_2 \rightsquigarrow E[t'_0] \ t_2 \quad (*) \\
 & \text{by ctx} \\
 & \text{with } E' = E \ t_2 \text{ and HPR} \\
 & E[t_0] \ t_2 \equiv E'[t_0] \rightsquigarrow E'[t'_0] \quad (*)
 \end{aligned}$$

6.2 Prova per definizione del lemma 2

$$\forall t, t'. \ t \rightsquigarrow t' \implies t \rightarrow t'$$

lemma a $\forall t, t'. t \rightarrow t' \implies E[t] \rightarrow E[t']$

lemma b $\forall t, t'. t \rightsquigarrow^P t' \implies t \rightarrow t'$

by inversion on HP	$t \equiv E[t_0]$	<i>HE0</i>
	$t' \equiv E[t'_0]$	<i>HE0'</i>
	$t_0 \rightsquigarrow^P t'_0$	<i>HPR</i>
by LB with HPR w.h.	$t_0 \rightarrow t'_0$	<i>HR</i>
by HE0, HE0' T.S.	$E[t_0] \rightarrow E[t'_0]$	
by LA with HR	the thesis holds	

Proof Lemma B Proof by case study on \rightsquigarrow^P

Proof Lemma A Proof by induction on E

- Base

$E = []$
 $TS t \rightarrow t'$ by HP

- Induzione.

- IH: $t \rightarrow t' \implies E'[t] \rightarrow E'[t']$
- $E = E'[t'']$
- by IH with HP. E' w.h. $E'[t] \rightarrow E'[t']$
- TS $(E' t'')[t] \rightarrow ()$

7 Simply Typed Lambda Calculus

I programmi descritti dal STLC sono un subset di tutti i programmi descritti dal ULC.

STLC non descrive però l'insieme di **tutti** i programmi che non falliscono. I *type system* fanno una over-approssimazione, rifiutando alcuni programmi che potrebbero ridurre a un valore.

In fine, un programma STLC può ancora divergere (finire in un loop infinito).

Programma ULC non STLC che non fallisce:

$(\lambda x.0)(\lambda y.3 + \lambda z.z)$

Il programma, assumendo call by name, riduce correttamente a 0. Questo è un comportamento che si può apprezzare a run time, ma non a compile time (dove vive il *type system*).

Tipi

$$\begin{array}{l} \tau := N \\ \tau \rightarrow \tau \end{array}$$

Judgment

vedi foto

recap

temini

$$\begin{array}{l} t := n \\ t \oplus t \\ \lambda x : \tau. t \\ x \\ t \ t \end{array}$$

v

$$\begin{array}{l} v := n \\ \lambda x : \tau. t \end{array}$$

tipi

$$\begin{array}{l} \tau := N \\ \tau \rightarrow \tau \end{array}$$

typing environment

$$\begin{array}{l} \Gamma := \emptyset \\ \Gamma, x : \tau \end{array}$$

8 Expanding The STLC

8.1 Aggiungere tuple

$$\begin{array}{l} t := \dots \\ | < t, t > \end{array}$$

$|t.1$
 $|t.2$

$\tau := \dots$
 $|\tau \times \tau$

$v := \dots$
 $|<v, v>$

$E := \dots$
 $|<E, t>$
 $|<v, E>$
 $|E.1$
 $|E.2$

$\frac{}{<v_1, v_2> .1 \rightsquigarrow^p v_1} p1 \frac{}{<v_1, v_2> .2 \rightsquigarrow^p v_2} p2$

8.2 Aggiungere inums

$t := \dots$
 $|inl\ t$
 $|inr\ t$
 $|case\ t\ of\ inl\ x \mapsto t | inr\ x \mapsto t$

$\tau := \dots$
 $|\tau_1 \cup + \tau_2$

$v := \dots$
 $|inl\ v$
 $|inr\ v$

$E := \dots$
 $|inl\ E$
 $|inr\ E$
 $|case\ t\ of\ inl\ x \mapsto t | inr\ x \mapsto t$

$$\frac{}{case\ inl\ v\ of\ inl\ x_1 \mapsto t_1 | inr\ x_2 \mapsto t_2 \rightsquigarrow^p t_1[v/x_1]} inL$$

$$\frac{}{case\ inr\ v\ of\ inl\ x_1 \mapsto t_1 | inr\ x_2 \mapsto t_2 \rightsquigarrow^p t_2[v/x_2]} inR$$

8.3 Booleani

Ci sono due modi in cui potremmo aggiungere booleani nel linguaggio.

- true: $\lambda x.\lambda y.x$
- false: $\lambda x.\lambda y.y$
- if t then t_1 else t_2 $t\ t_1\ t_2$

Questo fa evaluation sia di t_1 che t_2 . Possiamo risolvere così:

- true: $\lambda x.\lambda y..x\ 0$
- false: $\lambda x.\lambda y..y\ 0$
- if t then t_1 else t_2 $t\ (\lambda_.t_1)\ (\lambda_.t_2)$

Oppure così:

- true: $\lambda x.\lambda y..x$
- false: $\lambda x.\lambda y..y$
- if t then t_1 else t_2 $(t\ (\lambda_.t_1)\ (\lambda_.t_2))0$

$$\begin{array}{c}
\frac{\Gamma(x) = \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N} \quad \frac{\Gamma(y) = \mathbb{N} \rightarrow \mathbb{N} \quad \Gamma(a) = \mathbb{N}}{\Gamma \vdash y : \mathbb{N} \rightarrow \mathbb{N}}^{\text{var}} \quad \frac{\Gamma(a) = \mathbb{N}}{\Gamma \vdash a : \mathbb{N}}^{\text{var}}}{\Gamma \vdash x (y a) : \mathbb{N} \rightarrow \mathbb{N}}^{\text{val}} \quad \frac{\Gamma(y) = \mathbb{N} \rightarrow \mathbb{N} \quad \Gamma(b) = \mathbb{N}}{\Gamma \vdash y : \mathbb{N} \rightarrow \mathbb{N}}^{\text{var}} \quad \frac{\Gamma(b) = \mathbb{N}}{\Gamma \vdash b : \mathbb{N}}^{\text{var}}}{\Gamma \vdash x (y a) (y b) : \mathbb{N}}^{\text{app}} \\
\frac{\Gamma \left\{ \begin{array}{l} x : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}, \\ y : \mathbb{N} \rightarrow \mathbb{N}, \\ a : \mathbb{N}, \\ b : \mathbb{N} \end{array} \right. \vdash x (y a) (y b) : \mathbb{N}}{\Gamma \vdash \lambda b : \mathbb{N}. x (y a) (y b) : \mathbb{N} \rightarrow \mathbb{N}}^{\text{lam}} \\
\frac{\Gamma \vdash \lambda b : \mathbb{N}. x (y a) (y b) : \mathbb{N} \rightarrow \mathbb{N}}{\Gamma \vdash \lambda a : \mathbb{N}. \lambda b : \mathbb{N}. x (y a) (y b) : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})}^{\text{lam}} \\
\frac{\Gamma \vdash \lambda a : \mathbb{N}. \lambda b : \mathbb{N}. x (y a) (y b) : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})}{\Gamma \vdash \lambda x : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}. \lambda y : \mathbb{N} \rightarrow \mathbb{N}. \lambda a : \mathbb{N}. \lambda b : \mathbb{N}. x (y a) (y b) : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N}))}^{\text{lam}} \\
\frac{\Gamma \vdash \lambda x : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}. \lambda y : \mathbb{N} \rightarrow \mathbb{N}. \lambda a : \mathbb{N}. \lambda b : \mathbb{N}. x (y a) (y b) : (\mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N}))}{\Gamma \vdash \lambda x : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}. \lambda y : \mathbb{N} \rightarrow \mathbb{N}. \lambda a : \mathbb{N}. \lambda b : \mathbb{N}. x (y a) (y b) : (\mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}) \rightarrow ((\mathbb{N} \rightarrow \mathbb{N}) \rightarrow (\mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})))}^{\text{lam}}
\end{array}$$

$$\begin{array}{c}
\frac{}{x : \mathbb{N} \vdash 2 * x : \mathbb{N}}^{\text{num}} \\
\frac{}{\emptyset \vdash \lambda x : \mathbb{N}. 2 * x : \mathbb{N} \rightarrow \mathbb{N}}^{\text{lam}} \quad \frac{}{\emptyset \vdash 5 : \mathbb{N}}^{\text{num}} \\
\frac{}{\emptyset \vdash (\lambda x : \mathbb{N}. 2 * x) 5 : \mathbb{N}}^{\text{app}}
\end{array}$$

9 If Then Else

Assumiamo questo encoding per *true* e *false*:

$$\begin{aligned} True &= \text{inl}0 & Bool &= \mathbb{N} \uplus \mathbb{N} \\ False &= \text{inr}1 \end{aligned}$$

$$\text{if } t \text{ then } t' =$$

10 Properties of STLC

10.1 Type soundness

$$\begin{aligned} &\text{if } \emptyset \vdash t : \tau \text{ and } t \rightsquigarrow^* t' \text{ then either} \\ &\vdash t.VAL \\ &\text{or} \\ &\exists t'' . t' \rightsquigarrow t'' \end{aligned}$$

Se abbiamo un termine *well typed*, prima o poi riduce a un valore o a un termine che può ancora ridurre.

star-step.

$$\frac{}{t \rightsquigarrow^* t} \quad \frac{t \rightsquigarrow t'' \quad t'' \rightsquigarrow^* t'}{t \rightsquigarrow^* t'}$$

10.1.1 Progress

$$\begin{aligned} &\text{if } \emptyset \vdash t.\tau \text{ then either} \\ &\vdash t.VAL \text{ or} \\ &\exists t' . t \rightsquigarrow t' \end{aligned}$$

10.1.2 Preservation

$$\text{if } \emptyset \vdash t.\tau \text{ and } t \rightsquigarrow t' \text{ then } \emptyset \vdash t'.\tau$$

Lem: Canonicity

$$\begin{aligned} &\text{if } \Gamma \vdash v.N && \text{then } v = n \\ &\text{if } \Gamma \vdash v.\tau \rightarrow \tau' && \text{then } v = \lambda x : \tau. t' \\ &\text{if } \Gamma \vdash v.\tau \times \tau' && \text{then } v = \langle v_1, v_2 \rangle \\ &\text{if } \Gamma \vdash v.\tau \uplus \tau' && \text{then } v = \dots \end{aligned}$$

10.2 Normalization

if $\emptyset \vdash t.\tau$ then $\exists v.t \rightsquigarrow^ v$*

10.3 proofs

10.3.1 Proof of Progress

*if $\emptyset \vdash t.\tau$ then either
 $\vdash t.VAL$ or
 $\exists t'.t \rightsquigarrow t'$*

Proof by induction on the typing derivation.

Base

- t.VAR

$$\frac{\emptyset(x) = \tau}{\emptyset \vdash x.\tau} \text{contradiziona}$$

- t.NAT

$$\overline{\emptyset \vdash n.\mathbb{N}}$$

TS either $\vdash n.VAL$ or $\exists \tau'.n \rightsquigarrow t'$

Induction

- T-lam

$$\overline{\emptyset \vdash \lambda x : \tau. t' : \tau \rightarrow \tau'}$$

TS either $\vdash \lambda x : \tau. t.VAL$ or $\exists \dots$

- T-app

$$\frac{\emptyset \vdash t' : \tau' \rightarrow \tau \quad \emptyset \vdash t'' : \tau'}{\emptyset \vdash t' t'' : \tau}$$

10.3.2 Proof of Preservation

Assumendo $t \equiv E[t_0]$, abbiamo il judgment $\vdash E : \tau \rightarrow \tau$

$$\frac{\overline{\vdash [\cdot] : \tau \rightarrow \tau} \text{et-hole} \quad \vdash E : \tau \rightarrow (\tau'' \rightarrow \tau') \quad \emptyset \vdash t : \tau''}{\vdash E t : \tau \rightarrow \tau'} \text{et-app}$$

$$\begin{array}{c}
\frac{\emptyset \vdash (\lambda x : \tau. t) : \tau \rightarrow \tau' \quad \vdash E : \tau'' \rightarrow \tau}{\vdash (\lambda x : \tau. t)E : \tau'' \rightarrow \tau'} \text{et-lam} \\
\frac{\vdash E : \tau \rightarrow \mathbb{N} \quad \emptyset \vdash t : \mathbb{N}}{\vdash E \oplus t : \tau \rightarrow \mathbb{N}} \text{et-bopp} \\
\frac{\emptyset \vdash n : \mathbb{N} \quad \vdash E : \tau \rightarrow \mathbb{N}}{\vdash n \oplus E : \tau \rightarrow \mathbb{N}} \text{et-bopp}
\end{array}$$

Primitive Preservation *if $\emptyset \vdash t : \tau$ and $t \rightsquigarrow^P t'$ then $\emptyset \vdash t'.\tau$*

proof Casa analisys on \rightsquigarrow^P

Decomposition *if $\emptyset \vdash E[t] : \tau$ then $\exists \tau'. \vdash E : \tau' \rightarrow \tau$ and $\emptyset \vdash t : \tau'$*

Proof induction on E

Composition *if $\vdash E : \tau \rightarrow \tau'$ and $\emptyset \vdash t : \tau$ then $\emptyset \vdash E[t] : \tau'$*

Proof by induction on $\vdash E : \tau \rightarrow \tau'$

$$\begin{array}{ll}
\text{by inversion on } \text{HP}t \equiv E[t_0] & \text{HT0} \\
t' \equiv E[t'_0] & \text{HT1} \\
t_0 \rightsquigarrow^P t'_0 & \text{HTP} \\
\text{by HT0 to HP1 with } \emptyset \vdash E[t_0] : \tau & \text{HP1N} \\
\text{by HT1 to TH. TS} \emptyset \vdash E[t'_0] : \tau & \\
\text{by decomposition with HP1N w.h. } \vdash E : \tau' \rightarrow \tau & \text{HE} \\
\emptyset \vdash t_0 : \tau' & \text{HTT0} \\
\text{by prim. pres with HTT0 and HTP w.h. } \emptyset \vdash t'_0 : \tau' & \text{HTT1} \\
\text{by compos with HE and HTT1 W.h. } \emptyset \vdash E[t'_0] : \tau & \text{HF} \\
\text{by HF the thesis holds} &
\end{array}$$

10.3.3 Proof of Normalization

if $\emptyset \vdash t : \tau$ then $\exists v. t \rightsquigarrow^ v$*

Proof by induction on T.D of t

- base
- induction

$$- \quad t = t_1 \ t_2 \quad \frac{\emptyset \vdash t_1 : \tau' \rightarrow \tau \quad \emptyset \vdash t_2 : \tau'}{\emptyset \vdash t_1 \ t_2 : \tau}$$

Questo non possiamo provarlo con gli strumenti che abbiamo fin ora. Serve quindi introdurre le relazioni logiche.

11 Logical Relationships (and semantic typing)

$V[\tau]$ Quali valori costituiscono un tipo
 $E[\tau]$ Quali termini costituiscono un tipo
 $G[\Gamma]$ Sostituzione
 $\gamma ::= \emptyset$
 $|\gamma[v/x]$

Def SemTy (semantic typing) :

$$\Gamma \models t : \tau \hat{=} \forall \gamma \in G[\tau]. t\gamma \in E[\tau]$$

Semantic soundness

$$if \ \Gamma \vdash t : \tau \text{ then } \Gamma \models t : \tau$$

Se un programma è well typed in sintactic typing, lo è anche in semantic typing.

AAAH

$$\begin{aligned} V[\mathbb{N}] &= \{n\} \\ or \\ V[\mathbb{N}] &= \{v | v \equiv n\} \\ V[\tau' \rightarrow \tau] &= \{v | v \equiv \lambda x : \tau'. t \text{ and } \forall v' \text{ if } v' \in V[\tau'] \text{ then } t[v'/x] \in E[\tau]\} \\ E[\tau] &= \{t | \exists v. t \rightsquigarrow^* v \text{ and } v \in V[\tau]\} \\ V[\tau \times \tau'] &= \{v | v \equiv \langle v_1, v_2 \rangle \text{ and } t \in V[\tau] \text{ and } t' \in V[\tau']\} \\ V[\tau \uplus \tau'] &= \{v | v \equiv v.inlv_1 \text{ and } v_1 \in V[\tau]\} \cup \{v | v \equiv v.inrv_2 \text{ and } v_2 \in V[\tau']\} \end{aligned}$$

12 Proof of Normalization

proof by SS w.h. $\emptyset \models t.\tau$

...

first projection $t = t_1$

$$\Gamma \models \tau \times \tau' \text{ and}$$

13 lemma: vals in terms

$$\forall t \text{ if } t \in V[\tau] \text{ then } t \in E[\tau]$$

14 Compatibility lemmas

14.1 Application

$$\text{if } \Gamma \models t_1 : \tau \rightarrow \tau' \text{ and } \Gamma \models t_2 : \tau \text{ then } \Gamma \models t_1 \ t_2 : \tau'$$

proof

by def s.t take $\gamma \in G[\Gamma]$ t.s $(t_1 \ t_2)\gamma \in E[\tau']$

by def s.t with HP1 wh $t_1\gamma \in E[\tau \rightarrow \tau']$

by def $E \exists v_1. (t_1\gamma) \rightsquigarrow^* v_1$ and $v_1 \in V[\tau \rightarrow \tau']$

... by def $V \ v_1 \equiv \lambda x : \tau. t'_1$ and $\forall v'_1$ if $v'_1 \in V[\tau]$ then $t'_1[v'_1/x] \in E[\tau']$

by def s.t with HP2 wh $t_2\gamma \in E[\tau]$ by def $E \exists v_2. (t_2\gamma) \rightsquigarrow^* v_2$ and $v_2 \in V[\tau]$

$$(t_1 \ t_2)\gamma = (t_1\gamma)(t_2\gamma)$$

15 Introduction and Destruction

Le regole del linguaggio semantico possono essere divise in *introduzioni* e *eliminazioni*

$$\frac{\Gamma, x : \tau \models t : \tau'}{\Gamma \models \tau x : \tau t : \tau \rightarrow \tau'} \text{introduzione}$$

$$\frac{\Gamma \models t_1 : \tau \rightarrow \tau_1 \quad \Gamma \models t_2 : \tau}{\Gamma \models t_1 \ t_2 : \tau_1} \text{distruzione}$$

15.1 logica

$$\frac{A \quad A \Rightarrow B}{B} \Rightarrow E$$

[A]

\vdots

$\frac{B}{A \Rightarrow B} \Rightarrow I$

$$\frac{A \quad B}{A \wedge B} \wedge I$$

$$\frac{A \wedge B}{A} \text{AE1}$$

$$\frac{A \wedge B}{B} \text{AE2}$$

16 System F

$$t := \dots$$

$$|\Lambda\alpha.t$$

$$|t[\tau]$$

$$\tau := \dots$$

$$|\forall\alpha.\tau$$

$$|\alpha$$

$$v := \dots$$

$$|\Lambda\alpha.t$$

$$E := \dots$$

$$|E[\tau]$$

$$\Delta := \emptyset$$

$$|\Delta, \alpha$$

$$\Gamma := \emptyset$$

$$|\Gamma, x : \tau$$

$$\overline{(\Lambda\alpha t)[\tau] \rightsquigarrow^P t[\tau/\alpha]}^{big\beta}$$

Nuovo typing judgment:

$$\Delta, \Gamma \vdash t : \tau$$

Syntactic type checking:

$$\frac{\Delta}{\Delta, \Gamma \vdash \Delta\alpha t : \forall\alpha.\tau}$$

$$\frac{\overline{\Delta \vdash \mathbb{N}} \quad \Delta \vdash \tau \quad \Delta \vdash \tau'}{\Delta \vdash \tau \rightarrow \tau'}$$

...

16.1 Existential Types

Un record con almeno due label *is_oneis_off*. Definire il tipo *Switch* e un termine di questo tipo *jjk*: