Appunti Semantics

Diego Oniarti

Anno 2024-2025

Contents

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 metabariabile
- $:=$ 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

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

$$
\begin{aligned}
\overbrace{\Omega}^{\text{progrm state}} \quad ::= & t \\
& |fail
\end{aligned}
$$

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

Redexes

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

Values

$$
\begin{aligned}
v ::= & \ n \\
& |\lambda x.t
\end{aligned}
$$

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

rules

$$\frac{[n \oplus n'] = n''}{n \oplus n' \to n''} \quad \text{sos-bop}$$

$$\frac{}{(\lambda x.t)v \to t[\frac{v}{x}]} \quad \text{sos-beta}$$

$$\frac{t \to t''}{t \oplus t' \to t'' \oplus t'} \quad \text{sos-bop-1}$$

$$\frac{t \to t'}{n \oplus t \to n \oplus t'} \quad \text{sos-bop-2}$$

$$\frac{t \to t''}{t\ t' \to t''\ t'} \quad \text{sos-app-1}$$

$$\frac{t' \to t''}{(\lambda x.t)t' \to (\lambda x.t)\ t''} \quad \text{sos-app-2}$$

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 $\Omega \to \Omega$. 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 di può applicare alcuna regola.
Aggiungiamo quindi uno stato "*fail*" a Ω e delle regole per propagare questo fail.

Fails

$$\frac{}{(\lambda x.t) \oplus t \to fail} \text{ sos-f-L}$$

$$\frac{}{n\ t \to fail} \text{ sos-f-n}$$

$$\frac{}{n\ \oplus \lambda x.t \to fail} \text{ sos-f-L2}$$

$$\frac{t \to t'' \; t'' \to fail}{t \oplus t' \to fail} \text{ sos-bop-f1}$$

$$\frac{t \to t' \; t' \to fail}{n \oplus t \to fail} \text{ sos-bop-f2}$$

$$\frac{t \to t'' \; t'' \to fail}{t \; t' \to fail} \text{ sos-app-f1}$$

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

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:

$\frac{}{n \oplus n' \xrightarrow{N} n"}$	sos-bop N
$\frac{}{(\lambda x.t) \; t' \xrightarrow{N} t[\frac{t'}{x}]}$	sos-beta N
untouched	sos-app1N
untouched	sos-bop1N
untouched	sos-bop2N

4 Big Step

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

$$t \Downarrow v$$

Questo vuol dire che le inverence rules non fatto più pattern matching su $\Omega \to \Omega$ ma su $t \Downarrow v$ (il termine t riduce a un valore v).
rules:

$$\frac{}{v \Downarrow v} \text{ val}$$

$$\frac{t \Downarrow n \; t' \Downarrow n' \; n \oplus n' = n"}{t \oplus t' \Downarrow n"} \text{ bs-bop}$$

$$\frac{t \Downarrow \lambda x.t" \; t' \Downarrow v \; t"[v/x] \Downarrow v'}{t \; t' \Downarrow v'} \text{ bs-app}$$

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

$$\begin{aligned} &\Omega \rightsquigarrow \Omega && \text{main reduction} \\ &\Omega \rightsquigarrow^{\mathrm{p}} \Omega && \text{primitive reduction} \end{aligned}$$

$$\frac{t \rightsquigarrow^{\mathrm{p}} t'}{E[t] \rightsquigarrow E[t']}\ \text{ctx}$$

$$\frac{}{n \oplus n' \rightsquigarrow^{\mathrm{p}} n''}\ \text{c-bop}$$

$$\frac{}{(\lambda x.\, t)v \rightsquigarrow^{\mathrm{p}} t[v/x]}\ \text{c-beta}$$

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

$\forall t, t'.t \to t' \iff t \rightsquigarrow t'$
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 $\forall t, t'.t \to t' \implies t \rightsquigarrow t'$

lem.2 $\forall t, t'. t \to t' \iff t \rightsquigarrow t'$

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 stresso ragionamento vale.

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

- BOP

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

- BETA

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

Dimostriamo ora il passo induttivo per la prova del della 1:
In questo caso avremmo 4 casi induttivi da dimostrare (bop1, bop2, app1, app2) ma ne facciamo uno (app1) solo per brevità.

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

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

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

by ctx
with $E' = E\ t_2$ and HPR
$E[t_0]\ t_2 \equiv E'[t_0] \rightsquigarrow E'[t'_0]$ $^{(*)}$

6.2 Prova per definizione del lemma 2

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

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

lemma b $\forall t, t'.\ t \rightsquigarrow^{\mathrm{p}} t' \implies t \to t'$

by inversion on HP	$t \equiv E[t_0]$	$HE0$
	$t' \equiv E[t'_0]$	$HE0'$
	$t_0 \rightsquigarrow^{\mathrm{p}} t'_0$	HPR
by LB with HPR w.h.	$t_0 \to t'_0$	HR
by HE0,HE0' T.S.	$E[t_0] \to E[t'_0]$	
by LA with HR	the thesis holds	

Proof Lemma B Proof by case study on $\rightsquigarrow^{\mathrm{p}}$

Proof Lemma A Proof by induction on E

- Base

$$E = []$$
$$TSt \to t' \text{by HP}$$

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

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).

Progranna 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

$$\begin{aligned} \tau :=& N \\ & \tau \to \tau \end{aligned}$$

Judgment

$$vedi\ foto$$

recap

temini

$$\begin{aligned} t :=& n \\ & t \oplus t \\ & \lambda x : \tau .\, t \\ & x \\ & t\ t \end{aligned}$$

v

$$\begin{aligned} v :=& n \\ & \lambda x : \tau .\, t \end{aligned}$$

tipi

$$\begin{aligned} \tau :=& N \\ & \tau \to \tau \end{aligned}$$

typing environment

$$\begin{aligned} \Gamma :=& \emptyset \\ & \Gamma, x : \tau \end{aligned}$$

8 Expanding The STLC

8.1 Aggiungere tuple

$$\begin{aligned} t :=& \dots \\ & | < t, t > \\ & | t.1 \\ & | t.2 \end{aligned}$$

$$\begin{aligned} \tau :=& \dots \\ & | \tau \times \tau \end{aligned}$$

$$\begin{aligned} v :=& \dots \\ & | < v, v > \end{aligned}$$

$$\begin{aligned} E :=& \dots \\ & | < E, t > \\ & | < v, E > \\ & | E.1 \\ & | E.2 \end{aligned}$$

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

8.2 Aggiungere inums

$$\begin{aligned} t :=& \dots \\ & | inl\ t \\ & | inr\ t \\ & | case\ t\ of\ inl\ x \mapsto t | inr\ x \mapsto t \end{aligned}$$

$$\begin{aligned} \tau :=& \dots \\ & | \tau_1 \cup + \tau_2 \end{aligned}$$

$$\begin{aligned} v :=& \dots \\ & | inl\ v \\ & | inr\ v \end{aligned}$$

$$
\begin{aligned}
E := & \dots \\
& |inl\ E \\
& |inr\ E \\
& |case\ t\ of\ inl\ x \mapsto t | inr\ x \mapsto t
\end{aligned}
$$

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

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

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

$$
\dfrac{
  \dfrac{
    \dfrac{\Gamma(x) = \mathbb{N} \to \mathbb{N} \to \mathbb{N}}{\Gamma \vdash x : \mathbb{N} \to \mathbb{N} \to \mathbb{N}}\,\text{val}
    \quad
    \dfrac{
      \dfrac{\Gamma(y) = \mathbb{N} \to \mathbb{N}}{\Gamma \vdash y : \mathbb{N} \to \mathbb{N}}\,\text{var}
      \quad
      \dfrac{\Gamma(a) = \mathbb{N}}{\Gamma \vdash a : \mathbb{N}}\,\text{var}
    }{\Gamma \vdash y\ a : \mathbb{N}}\,\text{app}
  }{\Gamma \vdash x\ (y\ a) : \mathbb{N} \to \mathbb{N}}\,\text{app}
  \quad
  \dfrac{
    \dfrac{\Gamma(y) = \mathbb{N} \to \mathbb{N}}{\Gamma \vdash y : \mathbb{N} \to \mathbb{N}}\,\text{var}
    \quad
    \dfrac{\Gamma(b) = \mathbb{N}}{\Gamma \vdash b : \mathbb{N}}\,\text{var}
  }{\Gamma \vdash y\ b : \mathbb{N}}\,\text{app}
}{
  \dfrac{
    \Gamma \begin{cases} x{:}\mathbb{N}\to\mathbb{N}\to\mathbb{N}, \\ y{:}\mathbb{N}\to\mathbb{N}, \\ a{:}\mathbb{N}, \\ b{:}\mathbb{N} \end{cases} \vdash x\ (y\ a)\ (y\ b) : \mathbb{N}
  }{
    \dfrac{
      \begin{array}{l} x{:}\mathbb{N}\to\mathbb{N}\to\mathbb{N}, \\ y{:}\mathbb{N}\to\mathbb{N}, \\ a{:}\mathbb{N} \end{array} \vdash \lambda b : \mathbb{N}.\, x\ (y\ a)\ (y\ b) : \mathbb{N} \to \mathbb{N}
    }{
      \dfrac{
        \begin{array}{l} x{:}\mathbb{N}\to\mathbb{N}\to\mathbb{N}, \\ y{:}\mathbb{N}\to\mathbb{N} \end{array} \vdash \lambda a : \mathbb{N}.\, \lambda b : \mathbb{N}.\, x\ (y\ a)\ (y\ b) : \mathbb{N} \to (\mathbb{N} \to \mathbb{N})
      }{
        \dfrac{
          x : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \vdash \lambda y : \mathbb{N} \to \mathbb{N}.\, \lambda a : \mathbb{N}.\, \lambda b : \mathbb{N}.\, x\ (y\ a)\ (y\ b) : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to (\mathbb{N} \to \mathbb{N}))
        }{
          \emptyset : \lambda x : \mathbb{N} \to \mathbb{N} \to \mathbb{N}.\, \lambda y : \mathbb{N} \to \mathbb{N}.\, \lambda a : \mathbb{N}.\, \lambda b : \mathbb{N}.\, x\ (y\ a)\ (y\ b) : (\mathbb{N} \to \mathbb{N} \to \mathbb{N}) \to ((\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to (\mathbb{N} \to \mathbb{N})))
        }\,\text{lam}
      }\,\text{lam}
    }\,\text{lam}
  }\,\text{lam}
}\,\text{app}
$$

$$
\dfrac{
  \dfrac{
    \dfrac{}{x : \mathbb{N} \vdash 2 * x : \mathbb{N}}\,\text{num}
  }{\emptyset \vdash \lambda x : \mathbb{N}.\, 2 * x : \mathbb{N} \to \mathbb{N}}\,\text{lam}
  \quad
  \dfrac{}{\emptyset \vdash 5 : \mathbb{N}}\,\text{num}
}{\emptyset \vdash (\lambda x : \mathbb{N}.\, 2 * x)\ 5 : \mathbb{N}}\,\text{app}
$$

9 If Then Else

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

$$True = inl0 \qquad\qquad Bool = \mathbb{N} \uplus \mathbb{N}$$
$$False = inr1$$

$$if\ t\ then\ t' =$$

10 Properties of STLC

10.1 Type soundness

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

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

star-step.

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

10.1.1 Progress

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

10.1.2 Preservation

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

Lem: Canonicity

$$\begin{array}{lll} if\ \Gamma \vdash v.N & then & v = n\\ if\ \Gamma \vdash v.\tau \to \tau' & then & v = \lambda x : \tau.t'\\ if\ \Gamma \vdash v.\tau \times \tau' & then & v =< v_1, v_2 >\\ if\ \Gamma \vdash v.\tau \uplus \tau' & then & v = ... \end{array}$$

10.2 Normalization

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

10.3 proofs

10.3.1 Proof of Progress

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

Proof by induction on the typing derivation.

Base

- t.VAR

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

- t.NAT

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

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

Induction

- T-lam

$$\frac{}{\emptyset \vdash \lambda x : \tau. t' : \tau \to \tau'}$$

TS either $\underline{\vdash \lambda x : \tau. t.VAL}$ or $\exists$...

- T-app

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

10.3.2 Proof of Preservation

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

$$\frac{}{\vdash [\cdot] : \tau \to \tau}\text{et-hole}$$

$$\frac{\vdash E : \tau \to (\tau" \to \tau') \quad \emptyset \vdash t : \tau"}{\vdash E\ t : \tau \to \tau'}\text{et-app}$$

$$\frac{\emptyset \vdash (\lambda x : \tau . t) : \tau \to \tau' \quad \vdash E : \tau" \to \tau}{\vdash (\lambda x : \tau . t) E : \tau" \to \tau'} \text{et-lam}$$

$$\frac{\vdash E : \tau \to \mathbb{N} \quad \emptyset \vdash t : \mathbb{N}}{\vdash E \oplus t : \tau \to \mathbb{N}} \text{et-bopp}$$

$$\frac{\emptyset \vdash n : \mathbb{N} \quad \vdash E : \tau \to \mathbb{N}}{\vdash n \oplus E : \tau \to \mathbb{N}} \text{et-bopp}$$

Primitive Preservation $if\ \emptyset \vdash t : \tau\ and\ t \rightsquigarrow^{\mathrm{p}} t'\ then\ \emptyset \vdash t' . \tau$

proof Casa analisys on $\rightsquigarrow^{\mathrm{p}}$

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

Proof induction on E

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

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

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

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

$- \; t = t_1 \; t_2 \quad \dfrac{\emptyset \vdash t_1 : \tau' \to \tau \quad \emptyset \vdash t_2 : \tau'}{\emptyset \vdash t_1 \; t_2 : \tau}$

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

11 Logical Relationships (and semantic typing)

$$\begin{aligned}
&V\,[\tau]\text{Quali valori constituiscono un tipo}\\
&E\,[\tau]\text{Quali termini constituiscono un tipo}\\
&G\,[\Gamma]\text{Sostituzione}\\
&\gamma ::= \emptyset\\
&\quad\;\; |\gamma[v/x]
\end{aligned}$$

Def SemTy (semantic typing) :

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

Semantic soundness

$$if \;\Gamma \vdash t : \tau \; then \; \Gamma \vDash t : \tau$$

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

AAAH

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

12 Proof of Normalization

$$proof \; by \; SS \; w.h \; \emptyset \vDash t.\tau$$

$$...$$

first projection $t = t_1$

$$\Gamma \vDash \tau \times \tau' \; and$$

13 lemma: vals in terms

$$\forall t \; if \; t \in V[\tau] \; then \; t \in E[\tau]$$

14 Compatibility lemmas

14.1 Application

$$if \Gamma \vDash t_1 : \tau \to \tau' \; and \; \Gamma \vDash t_2 : \tau \; then \; \Gamma \vDash t_1 \; t_2 : \tau'$$

proof

$by \; def \; s.t \; take \; \gamma \in G[\Gamma] \; t.s \; (t_1 \; t_2)\gamma \in E[\tau']$
$by \; def \; s.t \; with \; HP1 \; wh \; t_1\gamma \in E[\tau \to \tau']$
$by \; def \; E \; \exists v_1.(t_1\gamma) \rightsquigarrow^* v_1 \; and \; v_1 \in V[\tau \to \tau']$
$\ldots by \; def \; V \; v_1 \equiv \lambda x : \tau. t_1' \; and \; \forall v_1'$ 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*

$$\frac{\Gamma, x : \tau \vDash t : \tau'}{\Gamma \vDash \tau x : \tau t : \underline{\tau \to \tau'}}\text{introduzione}$$

$$\frac{\Gamma \vDash t_1 : \underline{\tau \to \tau_1} \quad \Gamma \vDash t_2 : \tau}{\Gamma \vDash t_1 \; t_2 : \tau_1}\text{distruzione}$$

15.1 logica

$$\frac{A \quad A \implies B}{B} \implies \text{E}$$

$$\frac{\begin{matrix}[A] \\ \vdots \\ B\end{matrix}}{A \implies B} \implies \text{I}$$

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

$$\frac{A \land B}{A}\text{AE1}$$
$$\frac{A \land B}{B}\text{AE2}$$

16 System F

$$\begin{aligned} t ::= & \ldots \\ & |\Lambda\alpha.t \\ & |t[\tau] \end{aligned}$$

$$\begin{aligned} \tau ::= & \ldots \\ & |\forall\alpha.\tau \\ & |\alpha \end{aligned}$$

$$\begin{aligned} v ::= & \ldots \\ & |\Lambda\alpha.t \end{aligned}$$

$$\begin{aligned} E ::= & \ldots \\ & |E[\tau] \end{aligned}$$

$$\begin{aligned} \Delta ::= & \emptyset \\ & |\Delta, \alpha \end{aligned}$$

$$\begin{aligned} \Gamma ::= & \emptyset \\ & |\Gamma, x : \tau \end{aligned}$$

$$\frac{}{(\Lambda\alpha t)[\tau] \rightsquigarrow^{\mathrm{p}} t[\tau/\alpha]} big\beta$$

Nuovo typing judgment:

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

Syntactic type checking:

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

$$\frac{}{\Delta \vdash \mathbb{N}}$$

$$\frac{\Delta \vdash \tau \quad \Delta \vdash \tau'}{\Delta \vdash \tau \to \tau'}$$

...

16.1 Existential Types

Un record con almeno due label is_on e is_off. Definire il tipo Switch e un termine di questo tipo

17 free theorem

if

bool

$$\begin{aligned} \forall \alpha . \alpha \to \alpha \to \alpha \\ T : \Lambda\alpha . \lambda t : \alpha . \, \lambda f : \alpha . \, t \\ F : \Lambda\alpha . \lambda t : \alpha . \, \lambda f : \alpha . \, f \\ \text{if } v \text{ then } \ v_t \text{ else } v_f \equiv v[\tau] \ v_t \ v_f \end{aligned}$$

18 altro system F

$$pack \ \left\langle \mathbb{N}, \begin{cases} val = 0 \\ ison = \lambda x : \mathbb{N} . \, x == 0 \\ toggle = \lambda x : N . \ \text{if } x == 0 \text{ then } 1 \text{ else } 0 \end{cases} \right\rangle$$