

Appunti Semantics

Diego Oniarti

Anno 2024-2025

Contents

1	Lambda Calculus	2
1.1	Sintassi	3
2	SOS - Structural Operational Semantics	3
3	SOS - Call By Name	5
4	Big Step	6
4.1	Equivalenza con SS	6
5	Contextual Operation Semantics	6
5.1	COS, SS, CBV	6
6	Teorema di equivalenza SOS e COS	7
6.1	Prova per induzione del lemma 1	7
6.2	Prova per definizione del lemma 2	8
7	Simply Typed Lambda Calculus	9
8	Expanding The STLC	10
8.1	Aggiungere tuple	10
8.2	Aggiungere inums	11
8.3	Booleani	11
9	If Then Else	14
10	Properties of STLC	14
10.1	Type soundness	14
10.1.1	Progress	14
10.1.2	Preservation	14
10.2	Normalization	15
10.3	proofs	15
10.3.1	Proof of Progress	15
10.3.2	Proof of Preservation	15

10.3.3 Proof of Normalization	16
11 Logical Relationships (and semantic typing)	17
12 Proof of Normalization	17
13 lemma: vals in terms	18
14 Compatibility lemmas	18
14.1 Application	18
15 Introduction and Destruction	18
15.1 logica	18
16 System F	19
16.1 Existential Types	20
17 free theorem	20
18 altro system F	20
19 STLC-μ	20
19.1 isorecursive	21
19.2 equirecursive	21
19.3 Typing rule ISO	21
19.4 Modelliamo una lista in ISO	21
19.5 fold unfold cancellation	22
19.6 Diverging computation	22
19.7 recap isorecursion	22
19.8 Equirecursion	22
19.9 Logical relations	23
20 System F with recursive types	23
21 state	24
21.1 Adding Heap	24
22 higher order heap	25
23 Program Equivalence	25
23.1 Contextual equivalence	26
23.2 aaaa	26

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):

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

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

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

$$\begin{array}{c} \text{program state} \\ \widehat{\Omega} \end{array} ::= t \quad \text{or} \quad \text{fail}$$

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

Redexes

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

Values

$$v ::= n \quad \text{or} \quad \lambda x.t$$

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

rules

$$\begin{array}{ll} \frac{[n \oplus n'] = n''}{n \oplus n' \rightarrow n''} & \text{sos-bop} \\ \frac{(\lambda x.t)v \rightarrow t[\frac{v}{x}]}{(\lambda x.t)v \rightarrow t[\frac{v}{x}]} & \text{sos-beta} \\ \frac{t \rightarrow t''}{t \oplus t' \rightarrow t'' \oplus t'} & \text{sos-bop-1} \\ \frac{t \rightarrow t'}{n \oplus t \rightarrow n \oplus t'} & \text{sos-bop-2} \\ \frac{t \rightarrow t''}{t t' \rightarrow t'' t'} & \text{sos-app-1} \\ \frac{t' \rightarrow t''}{(\lambda x.t)t' \rightarrow (\lambda x.t) t''} & \text{sos-app-2} \end{array}$$

substitution

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

$$(t \oplus t')[v/x] = t[v/x] \oplus t'[v/x]$$

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

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

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

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 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:

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

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.

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

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

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

- BETA

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

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

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

6.2 Prova per definizione del lemma 2

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

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

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

$$\begin{array}{lll}
\text{by inversion on HP} & t \equiv E[t_0] & \text{HE0} \\
& t' \equiv E[t'_0] & \text{HE0'} \\
& t_0 \rightsquigarrow^P t'_0 & \text{HPR} \\
\text{by LB with HPR w.h.} & t_0 \rightarrow t'_0 & \text{HR} \\
\text{by HE0, HE0' T.S.} & E[t_0] \rightarrow E[t'_0] & \\
\text{by LA with HR} & \text{the thesis holds} &
\end{array}$$

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

Proof Lemma A Proof by induction on E

- Base

$$E = []$$

$$TSt \rightarrow t' \text{ by HP}$$

- 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

$$\tau := N$$

$$\tau \rightarrow \tau$$

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

$$\begin{array}{l} t := \dots \\ \mid < t, t > \\ \mid t.1 \\ \mid t.2 \end{array}$$

$$\begin{array}{l} \tau := \dots \\ \mid \tau \times \tau \end{array}$$

$$\begin{array}{l} v := \dots \\ \mid < v, v > \end{array}$$

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

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

8.2 Aggiungere inums

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

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

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

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

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

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}$$

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

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

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

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

11 Logical Relationships (and semantic typing)

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

Def SemTy (semantic typing) :

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

Semantic soundness

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

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

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

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

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

$$\frac{\begin{array}{c} [A] \\ \vdots \\ B \end{array}}{A \Rightarrow B} \Rightarrow I$$

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

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

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

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_on` e `is_off`. Definire il tipo `Switch` e un termine di questo tipo

17 free theorem

if

`bool`

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

18 altro system F

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

19 STLC- μ

STLC- μ aggiunge i tipi ricorsivi.

$$\tau ::= \dots \mid \mu \alpha. \tau$$

$$list[nat] \triangleq \mu \alpha. \underbrace{B}_{\text{empty}} \uplus (\mathbb{N} \times \alpha)$$

This unfolds to:

$$B \uplus (\mathbb{N} \times \mu\alpha. B \uplus (\mathbb{N} \times \alpha))$$

And we could keep unfolding the α over and over.

There are two schools of thought over this topic: isorecursive and equirecursive

19.1 isorecursive

We assume the folded and unfolded type are isomorphic. This isomorphism is seen at the term level.

$$\begin{aligned} t &::= \dots | fold_{\mu\alpha.\tau} t \\ &\quad | unfold_{\mu\alpha.\tau} t \\ v &::= \dots | fold_{\mu\alpha.\tau} t \end{aligned}$$

Questo metodo rende il type-checking deterministico, ma aggiunge uno step di riduzione

19.2 equirecursive

L'equirecursione rende il typing non deterministico ma non aggiunge step di riduzione.

È possibile dimostrare che i due metodi sono tecnicamente equivalenti.

19.3 Typing rule ISO

$$\frac{\Gamma \vdash t : \tau[\mu\alpha.\tau/\alpha]}{\Gamma \vdash fold_{\mu\alpha.\tau} t : \mu\alpha.\tau} \text{t-fold}$$

$$\frac{\Gamma \vdash t : \mu\alpha.\tau}{\Gamma \vdash unfold_{\mu\alpha.\tau} t : \tau[\mu\alpha.\tau/\alpha]} \text{t-unfold}$$

19.4 Modelliamo una lista in ISO

$$\begin{aligned} nil &\triangleq fold_{list[nat]} inl \ false \\ cons &\triangleq \lambda x : \mathbb{N}. \lambda l : list[nat]. fold_{list[nat]} inr \ \langle x, l \rangle \end{aligned}$$

typing derivation.

$$\emptyset \vdash \lambda x : \mathbb{N}. \lambda l : list[nat]. fold_{ln}$$

Couldn't be arsed. Look at the lecture.

List of generic α :

$$\forall \alpha. \mu \beta. B \uplus (\alpha \times \beta)$$

$$cons \triangleq \Lambda \beta. \lambda x : \beta. \lambda l : list[\alpha]. fold_{list[\alpha]} inr \langle x, l \rangle$$

19.5 fold unfold cancellation

19.6 Diverging computation

$$\begin{aligned} K &\triangleq \mu \alpha. \alpha \rightarrow \alpha \\ &(\lambda x. x \ x)(\lambda x. x \ x) \\ (\lambda x : \mu \alpha. \alpha \rightarrow \alpha. (unfold \ x) \ x) \ fold (\lambda x : \mu \alpha. \alpha \rightarrow \alpha. (unfold \ x) \ x) \end{aligned}$$

19.7 recap isorecursion

$$\begin{aligned} \tau &::= \dots | \mu \alpha. \tau \\ t &::= fold_{\mu \alpha. \tau} t | unfold_{\mu \alpha. \tau} t \\ unfold_{\mu \alpha. \tau} fold_{\mu \alpha. \tau} v &\rightsquigarrow^p v \\ \frac{\Delta; \Gamma \vdash t : \tau[\mu \alpha. \tau / \alpha]}{\Delta; \Gamma \vdash fold_{\mu \alpha. \tau} t : \mu \alpha. \tau} &\text{t-fold} \\ \frac{\Delta; \Gamma \vdash t : \mu \alpha. \tau}{\Delta; \Gamma \vdash unfold_{\mu \alpha. \tau} t : \tau[\mu \alpha. \tau / \alpha]} &\text{t-unfold} \end{aligned}$$

19.8 Equirecursion

$$\frac{\Delta, \Gamma \vdash t : \sigma \quad \Delta \vdash \sigma \overset{o}{=} \tau}{\Delta; \Gamma \vdash t : \tau} \text{t-eqi}$$

Questa regola può potenzialmente essere applicata in ogni passaggio del type-checking. Questo rende il processo non deterministico.

$$\Delta \vdash \sigma \overset{o}{=} \tau$$

$$\begin{aligned} \frac{\Delta \vdash \tau \overset{o}{=} \sigma}{\Delta \vdash \sigma \overset{o}{=} \tau} &\text{t-sym} \\ \frac{\Delta \vdash \sigma \overset{o}{=} \gamma \quad \Delta \vdash \gamma \overset{o}{=} \tau}{\Delta \vdash \sigma \overset{o}{=} \tau} &\text{t-trans} \end{aligned}$$

$$\begin{array}{c}
\frac{\tau \in \{\mathbb{N}, Bool, Unit\}}{\Delta \vdash \tau \doteq \tau} \text{t-base} \\
\frac{\Delta \vdash \tau_1 \doteq \sigma_1 \quad \Delta \vdash \tau_2 \doteq \sigma_2}{\Delta \vdash \tau_1 \star \tau_2 \doteq \sigma_1 \star \sigma_2} \text{t-bin} \\
\frac{\Delta, \alpha \vdash \tau \doteq \sigma}{\Delta \vdash \mu\alpha.\tau \doteq \mu\alpha.\sigma} \text{t-}\mu \\
\frac{\Delta \vdash \tau[\mu\alpha.t/\alpha] \doteq \sigma}{\Delta \vdash \mu\alpha.\tau \doteq \sigma} \text{t-unfold} \\
\frac{\alpha \in \Delta}{\Delta \vdash \alpha \doteq \alpha} \text{t-var}
\end{array}$$

19.9 Logical relations

La vecchia term relation

$$\mathcal{E}[\tau] = \{t \mid \underbrace{\exists v.t \rightsquigarrow^* v}_{\text{safety}} \text{ and } v \in \mathcal{V}[\tau]\}$$

aveva un certo concetto di "safety" che non accetta l'esecuzione divergente. Quindi dobbiamo modificarlo.

Introduciamo questo judgment $t \searrow_n t'$ chiamato *numbered stepping*. t steppa a t' in esattamente n step e non può più steppare.

La nuova definizione di safety che definiamo è questa:

$$\vdash t : \text{safe} \triangleq \forall k, t'. \text{ if } t \searrow_k t' \text{ then } \vdash t'.val$$

- $t \Downarrow$
- $t \not\rightarrow$
- $t \Uparrow$

La correttezza di questo viene dimostrata per induzione sulla k

20 System F with recursive types

$$\mathcal{V}[\tau]^\delta.\tau \times \delta \times v \times n$$

Ci aspettiamo che nella value relationship compaia un numero n , che indica per quanti step il termine è safe.

Modifichiamo *Semty* così:

$$\text{Semty}(\tau) = \{s \mid s \in \mathcal{P}(\mathbb{N} \times CVal(\tau)), \forall (k, v) \in S, \forall y < k, (j, v) \in S\}$$

$\mathcal{D}[\cdot] = \text{unchanged}$

$$G[\Gamma, x : \tau]^\delta = \{(k, \gamma[v/x]) \mid (k, \gamma) \in G[\Gamma]^\delta, (k, v) \in \mathcal{V}[\tau]^\delta\}$$

$$\mathcal{V}[\alpha]^\delta = \sigma(\alpha).S$$

$$\mathcal{V}[\mathbb{N}]^\delta = \{(k, n)\}$$

$$\mathcal{V}[\tau_1 \rightarrow \tau_2]^\delta = \{(k, \lambda x : \tau_1. t) \mid \forall j \leq k \forall v \text{ if } (j, v) \in \mathcal{V}[\tau]^\delta \text{ then } (j, t[v/x]) \in \mathcal{E}[\tau_2]^\delta\}$$

$$\mathcal{V}[\mu\alpha.\tau]^\delta = \{(k, fold_{\mu\alpha.\tau} v) \mid \forall j < k (j, v) \in \mathcal{V}[\tau[\mu\alpha.\tau/\alpha]]^\delta\}$$

$$\mathcal{E}[\tau]^\delta = \{(k, t) \mid \forall j < k, \forall t' \text{ if } t \searrow_j t' \text{ then } (k - j, t') \in \mathcal{V}[\tau]^\delta\}$$

$$\Delta, \Gamma \models t.\tau \triangleq \forall \sigma \in D[\Delta], \text{forall } (k, \gamma) \in G[\Gamma]^\delta, (k, t\gamma\delta) \in \mathcal{E}[\tau]^\delta$$

21 state

$f = ctr = 0 \lambda x : \mathbb{N}. \langle x * 2, ctr + 1 \rangle f \ 1; f \ 2$

Fino ad ora questo riduceva a $\langle 4, 1 \rangle$, perché non abbiamo stato.

21.1 Adding Heap

$$H ::= \emptyset \mid H, l \mapsto v$$

$$\Omega ::= H \triangleright t$$

$$t ::= \dots \mid \text{new } t \mid !t \mid t := t$$

$$\mathbb{E} ::= \dots \mid \text{new } \mathbb{E} \mid !\mathbb{E} \mid \mathbb{E} := t \mid l := \mathbb{E}$$

Dove $l \in \mathcal{L}$ è la "location", su cui non possiamo però fare pointer arithmetics o simili

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

$$\overline{H \triangleright \rightsquigarrow^p}^{\text{prm}}$$

nuove regole

$$\frac{fresh(l, H)}{H \triangleright \text{new } v \rightsquigarrow^p H; l \mapsto v \triangleright l}^{\text{nbv}}$$

$$\frac{H(l) = v}{H \triangleright !l \rightsquigarrow^p H \triangleright v}^{\text{read}}$$

$$\overline{H, l \mapsto v(l) = v}$$

$$\frac{\frac{H(l') = v}{H, l \mapsto v(l') = v}}{\frac{H' = H[l_1 \mapsto v/l_1 \mapsto \cdot]}{H \triangleright l_1 := vl \rightsquigarrow^p H' \triangleright \mathbf{0}} \text{write}}$$

Dove $\mathbf{0}$ è una costante generica. Potremmo usare *nil* o qualsiasi altra cosa.

22 higher order heap

$$\begin{array}{c} \tau ::= \dots | \text{ref } \tau \\ \frac{\Delta, \Gamma \vdash t : \tau}{\Delta, \Gamma \vdash \text{new } t : \text{ref } \tau} \text{t-new} \\ \frac{\Delta, \Gamma \vdash t : \text{ref } \tau}{\Delta, \Gamma \vdash !t : \tau} \text{t-read} \\ \frac{\Delta, \Gamma \vdash t : \text{ref } \tau \quad \Delta, \Gamma \vdash t : \tau}{\Delta, \Gamma \vdash t := t' : \mathbb{N}} \text{t-read} \end{array}$$

Aggiungiamo Σ per tenere traccia dei type bindings:

$$\begin{array}{c} \Sigma ::= \emptyset | \Sigma, l : \text{ref } \tau \\ \Delta, \Gamma \vdash t : \tau \rightarrow \Sigma, \Delta, \Gamma \vdash t : \tau \end{array}$$

Modifichiamo progress judgment

if $H : \Sigma$ and $\Sigma, \emptyset, \emptyset \vdash t : \tau$ then $\text{either } \vdash t.\text{val}$ or $\exists t'. H' : H \triangleright t \rightsquigarrow H' \triangleright t'$

Modifichiamo preservation:

if $H : \Sigma$ and $\Sigma, \emptyset, \emptyset \vdash t : \tau$ and $H \triangleright t \rightsquigarrow H' \triangleright t'$
then $\exists \Sigma' \supseteq \Sigma. H' : \Sigma'$ and $\Sigma', \emptyset, \emptyset \vdash t' : \tau$

dove:

$$\frac{\forall l : \text{ref } \tau \in \Sigma. \Sigma, \emptyset, \emptyset \vdash H(l) : \tau}{H : \Sigma} \quad \frac{\Sigma(l) = \text{ref } \tau}{\Sigma, \Delta, \Gamma \vdash l : \text{ref } \tau}$$

23 Program Equivalence

Due programmi sono equivalenti se un osservatore esterno non può in alcun modo distinguere i due, con le limitazioni del linguaggio utilizzato.

23.1 Contextual equivalence

Program contexts are a generalization of evaluation context but capture a different concept.

Program contexts represent the resto of the world we're linking with.

The goal of program context is to divide the program in two different parts. There's the one we care about, which gets plugged in the hole, and the "observer" which has the hole.

Grammar of program context

$$\begin{aligned}
C ::= & [\cdot] \\
& |\lambda x : \tau. C \\
& | C \ t \\
& | t \ C \\
& | t \oplus C | C \oplus t \\
& | C.1 | C.2 \\
& | \langle C, t \rangle | \langle t, C \rangle \\
& | \text{case } t \text{ of } \begin{cases} \text{inl } x \mapsto C \\ \text{inr } x \mapsto t \end{cases} \\
& | \text{case } t \text{ of } \begin{cases} \text{inl } x \mapsto t \\ \text{inr } x \mapsto C \end{cases} \\
& | \text{case } C \text{ of } \begin{cases} \text{inl } x \mapsto t \\ \text{inr } x \mapsto t \end{cases}
\end{aligned}$$

huh?

$$\begin{aligned}
t_1 \simeq t_2 & \triangleq \forall C. C[t_1] \Downarrow \iff C[t_2] \Downarrow \\
t \Downarrow & \triangleq \exists n, v. t \rightsquigarrow^n v
\end{aligned}$$

Possiamo raffinare la definizione aggiungendo il typing:

$$\begin{aligned}
\emptyset \vdash t_1 \simeq t_2 : \tau & \triangleq \forall C \\
& \text{if } C\emptyset : \tau \rightarrow \Gamma, \tau' \text{ and } \emptyset \vdash t_1 : \tau \text{ and } \emptyset \vdash t_2 : \tau \\
& \text{then } C[t_1] \Downarrow \iff C[t_2] \Downarrow
\end{aligned}$$

23.2 aaaa

$$\begin{aligned}
\Delta; \Gamma \vdash t_1 \approx t_2 : \tau & \triangleq \forall \delta \in D[\Delta]. \forall (\gamma_1, \gamma_2) \in G[\Gamma]^\delta. (t_1 \gamma_1 \delta, t_2 \gamma_2 \delta) \in \mathcal{E}[\tau]^\delta \\
\delta & ::= \emptyset | \delta, \alpha \mapsto (\tau_1 S_1, \tau_2 S_2) \\
\mathcal{E}[\tau]^\delta & = \{(t_1, t_2) | \exists v_1, v_2. \text{ if } t_1 \rightsquigarrow^* v_1 \text{ and } t_2 \rightsquigarrow^* v_2 \text{ then } (v_1, v_2) \in \mathcal{V}[\tau]^\delta\}
\end{aligned}$$

$$\begin{aligned}
\mathcal{V}[\mathbb{N}]^\delta &= \{(n, n)\} \\
\mathcal{V}[\tau \rightarrow \tau']^\delta &= \{(\lambda x : \tau. t_1, \lambda x : \tau. t_2) \mid \forall v_1, v_2 \text{ if } (v_1, v_2) \in \mathcal{V}[\tau]^\delta \text{ then } (t_1[v_1/x], t_2[v_2/x]) \in \mathcal{E}[\tau']^\delta\} \\
\mathcal{V}[\alpha]^\delta &= \delta(\alpha) \\
\mathcal{V}[\forall \alpha. \tau]^\delta &= \{(\Lambda \alpha. t_1, \Lambda \alpha. t_2) \mid \forall \tau_1, \tau_2, \forall S \in VRel(\tau_1, \tau_2). (t[\tau_1/\alpha, t_2[\tau_2/\alpha]]) \in \mathcal{E}[\tau]^\delta, \alpha \mapsto (\tau_1, \tau_2); S\} \\
VRel(\tau, \tau') &= \mathcal{P}(CVal(\tau) \times CVal(\tau')) \\
\mathcal{V}[\exists \alpha. \tau]^\delta &= \{(pack \ \langle \tau_1, v_2 \rangle \text{ as } \exists \alpha. \tau, pack \ \langle \tau_2, v_2 \rangle \text{ as } \exists \alpha. \tau) \mid \exists S \in VRel(\tau_1, \tau_2). (v_1, v_2) \in \mathcal{V}[\tau]^\delta, \alpha \mapsto (\tau_1, \tau_2); S\}
\end{aligned}$$

Generalizziamo la term relation e ne aggiungiamo una context

$$\begin{aligned}
\mathcal{E}[\tau]^\delta &= \{(t_1, t_2) \mid \forall E_1, E_2. \text{ if } (E_1, E_2) \in K[\tau]^\delta \text{ then } E_1[t_1] \Downarrow \iff E_2[t_2] \Downarrow\} \\
K[\tau]^\delta &= \{(E_1, E_2) \mid \forall v_1, v_2 \text{ if } (v_1, v_2) \in \mathcal{V}[\tau]^\delta \text{ then } E[v_1] \Downarrow \iff E[v_2] \Downarrow\}
\end{aligned}$$