1 [Win93]: integrazioni

1.1 [Win93, Ch 5]: semantica operazione con errore per IMP

In questa semantica operazionale una locazione puo' contenere un valore (intero) o essere indefinite (e quindi un tentativo di leggerne il contenuto e' considerato un errore). Le categorie sintattiche sono invariate

$$\begin{array}{rcl} n \in \mathsf{N} & ::= & \ldots \\ x \in \mathsf{Loc} & ::= & \ldots \\ a \in \mathsf{Aexp} & ::= & n \mid x \mid a_1 + a_2 \mid \ldots \\ b \in \mathsf{Bexp} & ::= & \mathsf{true} \mid \mathsf{false} \mid a_1 = a_2 \mid b_1 \wedge b_2 \mid \ldots \\ c \in \mathsf{Cmd} & ::= & x{:=}\, a \mid \mathsf{skip} \mid c_1; c_2 \mid \mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2 \mid \mathsf{while}\ b\ \mathsf{do}\ c \end{array}$$

I domini semantici e le asserzioni usate nella semantica operazionale con errore cambiano. Uno stato e' una funzione parziale a dominio finito $s \in \mathsf{S} \triangleq \mathsf{Loc} \stackrel{fin}{\to} \mathsf{N}$, mentre alle asserzioni per la valutazione di una espressione ($a, s \to n$ e $b, s \to t$) e l'esecuzione di un comando ($s, c \to s'$) vanno aggiunte le asserzioni per indicare un errore ($a, s \to err$ e $b, s \to err$ e $c, s \to err$).
Le regole per definire la semantica operazionale sono estese per tener conto della possibilita' di errore. Riportiamo completamente le regole per valutare le espressioni aritimiche, mentre per le altre categorie sintattiche indichiamo solo le nuove regole (per la propagazione degli errori):

- $a, s \to n \mid err$

  $$n\ \frac{}{n, s \to n} \qquad x\ \frac{}{x, s \to n}\ n = s(x) \qquad +\ \frac{a_1, s \to n_1 \qquad a_2, s \to n_2}{a_1 + a_2, s \to n}\ n = n_1 + n_2$$

  $$x.\mathrm{err}\ \frac{}{x, s \to err}\ x \notin dom(s) \qquad +.\mathrm{err}.1\ \frac{a_1, s \to err}{a_1 + a_2, s \to err} \qquad +.\mathrm{err}.2\ \frac{a_1, s \to n_1 \qquad a_2, s \to err}{a_1 + a_2, s \to err}$$

  La regola (x.err) e' detta di **error generation**, mentre (+.err.1) e (+.err.2) sone dette di **error propagation**.

- $b, s \to n \mid err$

  $$=.\mathrm{err.i}\ \frac{}{a_1 = a_2, s \to err}\ a_i, s \to err \qquad \wedge.\mathrm{err}.1\ \frac{b_1, s \to err}{b_1 \wedge b_2, s \to err} \qquad \wedge.\mathrm{err}.2\ \frac{b_1, s \to t \qquad b_2, s \to err}{b_1 \wedge b_2, s \to err}$$

  si possono introdurre regole di propagazione diverse, in corrispondenza delle diverse strategie di valutazione delle espressioni booleane.

- $c, s \to s' \mid err$, useremo la metavariabile r per indicare un possibile risultato (cioe' s' o err)

  $$:=.\mathrm{err}\ \frac{}{x{:=}\, a, s \to err}\ a, s \to err \qquad ;.\mathrm{err}.1\ \frac{c_1, s \to err}{c_1; c_2, s \to err} \qquad ;.\mathrm{err}.2\ \frac{c_1, s \to s' \qquad c_2, s' \to err}{c_1; c_2, s \to err}$$

  $$\mathrm{if.err}\ \frac{}{\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, s \to err}\ b, s \to err \qquad \mathrm{while.err}\ \frac{}{\mathsf{while}\ b\ \mathsf{do}\ c, s \to err}\ b, s \to err$$

  $$\mathrm{if.true}\ \frac{c_1, s \to r}{\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, s \to r}\ b, s \to \mathsf{true} \qquad \mathrm{if.false}\ \frac{c_2, s \to r}{\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, s \to r}\ b, s \to \mathsf{false}$$

  $$\mathrm{while.true.err}\ \frac{c, s \to err}{\mathsf{while}\ b\ \mathsf{do}\ c, s \to err}\ b, s \to true \qquad \mathrm{while.true}\ \frac{c, s \to s' \qquad \mathsf{while}\ b\ \mathsf{do}\ c, s \to r}{\mathsf{while}\ b\ \mathsf{do}\ c, s' \to r}\ b, s \to true$$

Anche per questa semantica si puo' dimostrare la univocita' del risultato

Proposition 1.1 (Univocita') *Se $c, s \to r_1$ e $c, s \to r_2$, allora $r_1 = r_2$.*

1.2 [Win93, Ch 5]: analisi statica per IMP

Descriviamo una semplice analisi statica per IMP, che garantisce (sotto certe ipotesi sul'insieme delle locazioni che contengono un valore) l'assenza di errori a tempo di esecuzione. Tale analisi si puo' descrivere mediante regole per derivare asserzioni della seguente forma, dove $X \subseteq_{fin} \mathsf{Loc}$:

- $a/b, X \to ok$, cioe' se le locazioni in X sono definite, allora la valutazione di a/b non genera errore

- $c, X \to X'$, cioe' se le locazioni in X sono definite, allora l'esecuzione di c non genera errore e nello stato finale (se esiste) tutte le locazioni in X' saranno definite

Ci limitiamo a dare le regole per derivare asserzioni della forma $c, X \to X'$. La caratteristica essenziale di queste regole e' che sono ben fondate (rispetto alla relazione di sottocomando immediato):

- $:= \ \dfrac{}{x := a, X \to X \cup \{x\}}\ a, X \to ok$ $\quad$ skip $\dfrac{}{\mathsf{skip}, X \to X}$ $\quad$; $\dfrac{c_1, X \to X' \quad c_2, X' \to X''}{c_1; c_2, X \to X''}$

  if $\dfrac{c_1, X \to X_1 \quad c_2, X \to X_2}{\mathsf{if}\ b\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, X \to X_1 \cap X_2}\ b, X \to ok$ $\quad$ while $\dfrac{c, X \to X'}{\mathsf{while}\ b\ \mathsf{do}\ c, X \to X}\ b, X \to ok$

Proposition 1.2 (Proprieta' dell'analisi statica)

- *se* $c, X \to X'$, *allora* $X \subseteq X'$
- *per ogni* $X \subseteq_{fin} \mathsf{Loc}$ *esiste* $X' \subseteq_{fin} \mathsf{Loc}$ *t.c.* $c, X \to X'$
- *se* $c, X \to X_1$ *e* $c, X \to X_2$, *allora* $X_1 = X_2$

Proof Per induzione sulla struttura di c. ∎

La relazione tra semantica operazione con errore e analisi statica e' data da

Theorem 1.3 (Correttezza)

- *se* $a, s \to r$ *e* $a, X \to ok$ *e* $X \subseteq dom(s)$, *allora* r *e' un intero*
- *se* $b, s \to r$ *e* $b, X \to ok$ *e* $X \subseteq dom(s)$, *allora* r *e' un valore booleano*
- *se* $c, s \to r$ *e* $c, X \to X'$ *e* $X \subseteq dom(s)$, *allora* r *e' un stato* s' *t.c.* $X' \subseteq dom(s')$

Proof Per induzione sulla derivazione di un asserto della semantica operazione con errore. ∎

1.3 [Win93, Ch 5]: osservazioni e computational adequacy

Una richiesta minimale su qualunque semantica per un linguaggio di programmazione, e' la possibilita' di descrivere il **comportamente osservabile** dei programmi (che nel caso di IMP identificheremo con i comandi). L'osservazione piu' elementare che si puo' immaginare e' la **terminazione** dell'esecuzione di un programma. Vediamo come l'osservazione "c termina" puo' essere definita formalmente nelle varie semantiche date per IMP

1. semantica operazionale a grandi passi: $\exists s \in S.c, s_0 \to s$, dove s_0 e' uno stato iniziale prefissato, p.e. lo stato dove tutte le locazioni contengono 0
2. semantica operazionale a piccoli passi: $\exists s \in S.c, s_0 \to_1^* s$, dove s_0 e' come sopra
3. semantica denotazionale: $[\![c]\!](s_0)$ e' definita
4. semantica operazionale con errore: $\exists r \in S \cup \{err\}.c, s_0 \to r$, dove la scelta piu' naturale per lo stato iniziale s_0 e' quello dove tutte le locazioni sono indefinite.

Quindi ciascuna delle 4 semantiche identifica un sottoinsieme O_i delle possibili osservazioni $\{$"c termina"$|c \in \mathsf{Cmd}\}$. Quando due semantiche identificano lo stesso sottoinsieme di osservazioni si parla di **computational adequacy**. Si puo' dimostrare che le prime tre semantiche identificano lo stesso insieme di osservazioni della forma "c termina".
Vi sono altre osservazioni che potrebbero essere interessanti, p.e.

- "c termina in n passi", tale osservazione e' definibile come $\exists s \in S.c, s_0 \to_1^n s$ usando la semantica operazionale a piccoli passi, ma non lo e' nelle altre sematiche, perche' sono "troppo astratte"
- "c termina con errore", tale osservazione e' definibile come $c, s_0 \to err$ usando la semantica operazionale con errore, mentre non ha senso nelle altre sematiche (dove tutte le locazione sono definite)
- "c termina senza errore", tale osservazione e' definibile come $\exists s \in S.c, s_0 \to s$ usando la semantica operazionale con errore, mentre nelle altre sematiche coincide con "c termina" (poiche' non e' possibile avere errori).

1.4 [Win93, Ch 5]: osservazioni ed equivalenza osservazionale

Fissato l'insieme delle osservazioni O, p.e. la terminazione, e l'insieme P dei programmi su cui effettuare le osservazione, ogni semantica indurra' una relazione $p \models o$ "il programma p soddisfa l'osservazione o", p.e. la semantica operazionale a grandi passi identifica univocamente (fissato s_0) il sottoinsieme dei programmi che terminano. A partire da questi dati, cioe' gli insiemi O e P e la relazione $\models$, possiamo definire una equivalenza tra *frammenti di programma* che catturi l'idea intuitiva di indistinguibilita' rispetto alle osservazioni. Per far questo e' necessario anzitutto fissare per quali frammenti ha senso chiedere se sono indistinguibili, p.e. nel caso di IMP ha senso confrontare tra loro le espressioni aritmetiche, ma non ha senso confrontare una espressione aritmetica ed un comando. Quindi, per ogni *tipo* τ di frammento dobbiamo dare un insieme E_τ di *frammenti* di quel tipo, ed un insieme C_τ di *contesti* per frammenti di tipo τ. Scriveremo $C[\Box_\tau]$ per un elemento di C_τ. Il requisito minimale su E_τ e C_τ e' una operazione per inserire un frammento $e \in E_\tau$ in un contesto $C[\Box_\tau] \in C_\tau$ in modo da ottenere un programma $C[e] \in P$. L'**equivalenza osservazionale** $\approx_\tau$ su E_τ e' data da $e_1 \approx_\tau e_2 \overset{\Delta}{\iff} \forall C[\Box_\tau] \in C_\tau.\forall o \in O.C[e_1] \models o \iff C[e_2] \models o$ (si puo' definire anche un pre-ordine osservazionale, sostituendo $\iff$ con $\Longrightarrow$).

Si puo' dimostrare che nel caso della terminazione definita in termini della semantica operazione a grandi passi per IMP, sebbene essa dipenda dalla scelta dello stato iniziale s_0, l'equivalenza osservazionale indotta non cambia al variare di s_0. D'altra parte, si puo' dimostrare (sempre nel caso di IMP), che l'equivalenza osservazionale indotta dalla terminazione definita in termini della semantica operazione con errore distingue di piu' di quella indotta dalla semantica operazionale a grandi passi, p.e. $\mathsf{skip} \approx_c x := x$ e' falsa nel primo caso e vera nel secondo.

In generale, una semantica denotazionale induce una **equivalenza denotazionale** $e_1 =_\tau e_2 \overset{\Delta}{\iff} [\![e_1]\!] = [\![e_2]\!]$, cioe' due frammenti sono equivalenti se hanno la stessa interpretazione. E' naturale chiedersi in che relazione stanno $=_\tau$ e $\approx_t$ (nel caso di interpretazione in cpo e' ragionevole confrontare anche l'ordine parziale con il pre-ordine osservazionale). Si ha che $e_1 =_\tau e_2$ implica $e_1 \approx_\tau e_2$, sotto ipotesi ragionevoli

- composizionalita' per i contesti $C[\Box_\tau] \in C_\tau$, cioe' $\forall e_1, e_2 \in E_\tau.[\![e_1]\!] = [\![e_2]\!] \Longrightarrow [\![C[e_1]]\!] = [\![C[e_2]]\!]$
- composizionalita' per le osservazioni, cioe' $\forall o \in O.\forall p_1, p_2 \in P.[\![p_1]\!] = [\![p_2]\!] \wedge p_1 \models o \Longrightarrow p_2 \models o$

che in generale seguono dal principio di *composizionalita'* "l'interpretazione di una espressione e' funzione delle interpretazioni delle sue sottoespressioni (immediate)".

1.5 [Win93, Ch 11]: semantica operazionale e type system per HOFL

Le categorie sintattiche sono invariate (a parte il costrutto per la ricorsione)

$$\begin{array}{rcl}
n \in \mathsf{N} & ::= & \ldots \\
x \in \mathsf{Var} & ::= & \ldots \\
t \in \mathsf{Exp} & ::= & n \mid t_1 + t_2 \mid \mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \mid (t_1, t_2) \mid \mathsf{fst}(t) \mid \mathsf{snd}(t) \mid x \mid \lambda x.t \mid t_1\ t_2 \mid \mathsf{rec}\ x.t \mid \mathsf{let}\ x = t_1\ \mathsf{in}\ t_2 \\
\tau \in \mathsf{Type} & ::= & \mathsf{int} \mid \tau_1 * \tau_2 \mid \tau_1 \to \tau_2
\end{array}$$

Supponiamo date le definizioni di

- insieme $\mathrm{FV}(t) \subseteq_{fin} \mathsf{Var}$ delle variabili libere di t
- sostituzione (senza renaming delle variabili legate) $t'[x := t]$ di x con t in t' [La stessa notazione e' usata per indicare la modifica di una funzione in un punto]

Indichiamo con Exp_0 l'insieme dei $t \in \mathsf{Exp}$ senza variabili libere.

La semantica operazione CBV (call-by-value, detta anche eager) e' data mediante regole per derivare asserzioni della forma $t \to v$ con $t \in \mathsf{Exp}_0$ e $v \in \mathsf{Val}_0$, dove $v \in \mathsf{Val} ::= n \mid (v_1, v_2) \mid \lambda x.t$

- $n\ \dfrac{}{n \to n} \qquad + \ \dfrac{t_1 \to n_1 \quad t_2 \to n_2}{t_1 + t_2 \to n}\ n = n_1 + n_2$

  $\text{if.true}\ \dfrac{t_0 \to 0 \quad t_1 \to v}{\mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \to v} \qquad \text{if.false}\ \dfrac{t_0 \to n \quad t_2 \to v}{\mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \to v}\ n \neq 0$

- $\text{pair}\ \dfrac{t_1 \to v_1 \quad t_2 \to v_2}{(t_1, t_2) \to (v_1, v_2)} \qquad \text{fst}\ \dfrac{t \to (v_1, v_2)}{\mathsf{fst}(t) \to v_1} \qquad \text{snd}\ \dfrac{t \to (v_1, v_2)}{\mathsf{snd}(t) \to v_2}$

- $\lambda \dfrac{}{\lambda x.t \to \lambda x.t}$ app $\dfrac{t_1 \to \lambda x.t \quad t_2 \to v \quad t[x:=v] \to v'}{t_1\ t_2 \to v'}$ rec $\dfrac{t[x:=\mathsf{rec}\ x.t] \to v}{\mathsf{rec}\ x.t \to v}$

  let $\dfrac{t_1 \to v \quad t_2[x:=v] \to v'}{\mathsf{let}\ x = t_1\ \mathsf{in}\ t_2 \to v'}$

La semantica operazione CBN (call-by-name, detta anche lazy) e' data mediante regole per derivare asserzioni della forma $t \to v$ con $t \in \mathsf{Exp}_0$ e $v \in \mathsf{Val}_0$, dove $v \in \mathsf{Val} ::= n \mid (t_1, t_2) \mid \lambda x.t$. Riportiamo le sole regole che differiscono da quelle date per la semantica operazione CBV

- pair $\dfrac{}{(t_1,t_2) \to (t_1,t_2)}$ fst $\dfrac{t \to (t_1,t_2) \quad t_1 \to v_1}{\mathsf{fst}(t) \to v_1}$ snd $\dfrac{t \to (t_1,t_2) \quad t_2 \to v_2}{\mathsf{snd}(t) \to v_2}$
- app $\dfrac{t_1 \to \lambda x.t \quad t[x:=t_2] \to v}{t_1\ t_2 \to v}$ let $\dfrac{t_2[x:=t_1] \to v}{\mathsf{let}\ x = t_1\ \mathsf{in}\ t_2 \to v}$

Il type system si puo' descrivere mediante regole per derivare asserzioni della seguente forma, dove $\Gamma\colon \mathsf{Var} \stackrel{fin}{\to} \mathsf{Type}$:

- $\Gamma \vdash t{:}\tau$, t ha tipo τ nel typing context Γ.

Le regole del type system sono ben fondate (rispetto alla relazione di sottotermine immediato), tuttavia non vale l'unicita' del tipo (fissati Γ e t):

- $n \dfrac{}{\Gamma \vdash n{:}\,\mathsf{int}}$ $+ \dfrac{\Gamma \vdash t_1{:}\,\mathsf{int} \quad \Gamma \vdash t_2{:}\,\mathsf{int}}{\Gamma \vdash t_1 + t_2{:}\,\mathsf{int}}$ if $\dfrac{\Gamma \vdash t_0{:}\,\mathsf{int} \quad \Gamma \vdash t_1{:}\,\tau \quad \Gamma \vdash t_2{:}\,\tau}{\Gamma \vdash \mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2{:}\,\tau}$

  pair $\dfrac{\Gamma \vdash t_1{:}\,\tau_1 \quad \Gamma \vdash t_2{:}\,\tau_2}{\Gamma \vdash (t_1,t_2){:}\,\tau_1 * \tau_2}$ fst $\dfrac{\Gamma \vdash t{:}\,\tau_1 * \tau_2}{\Gamma \vdash \mathsf{fst}(t){:}\,\tau_1}$ snd $\dfrac{\Gamma \vdash t{:}\,\tau_1 * \tau_2}{\Gamma \vdash \mathsf{snd}(t){:}\,\tau_2}$
- $x \dfrac{}{\Gamma \vdash x{:}\,\tau}\ \Gamma(x) = \tau$ $\lambda \dfrac{\Gamma[x:=\tau_1] \vdash t{:}\,\tau_2}{\Gamma \vdash \lambda x.t{:}\,\tau_1 \to \tau_2}$ app $\dfrac{\Gamma \vdash t_1{:}\,\tau_1 \to \tau_2 \quad \Gamma \vdash t_2{:}\,\tau_1}{\Gamma \vdash t_1\ t_2{:}\,\tau_2}$

  rec $\dfrac{\Gamma[x:=\tau] \vdash t{:}\,\tau}{\Gamma \vdash \mathsf{rec}\ x.t{:}\,\tau}$ let $\dfrac{\Gamma \vdash t_1{:}\,\tau_1 \quad \Gamma[x:=\tau_1] \vdash t_2{:}\,\tau_2}{\Gamma \vdash \mathsf{let}\ x = t_1\ \mathsf{in}\ t_2{:}\,\tau_2}$

Proposition 1.4 *Le seguenti regole sono ammissibile:*

weakening $\dfrac{\Gamma \vdash t{:}\,\tau}{\Gamma' \vdash t{:}\,\tau}\ \Gamma \subseteq \Gamma'$ subst $\dfrac{\emptyset \vdash t{:}\,\tau \quad \Gamma[x:=\tau] \vdash t'{:}\,\tau'}{\Gamma \vdash t'[x:=t]{:}\,\tau'}$

Proof (weakening) si dimostra per induzione sulla derivazione di $\Gamma \vdash t{:}\,\tau$. (subst) si dimostra per induzione sulla derivazione di $\Gamma[x:=\tau] \vdash t'{:}\,\tau'$, e nel caso $t' \equiv x$ si usa (weakening). ∎

Theorem 1.5 (Type preservation) *Se $t \to v$ e $\emptyset \vdash t{:}\,\tau$, allora $\emptyset \vdash v{:}\,\tau$.*

Proof Per induzione sulla derivazione di $t \to v$. Nei casi $t \equiv t_1\ t_2 \mid \mathsf{rec}\ x.t \mid \mathsf{let}\ x = t_1\ \mathsf{in}\ t_2$ si usa (subst). ∎

Per HOFL si puo' dare una semantica operazionale con errore (in analogia con IMP), e rafforzare il precedente risultato

Theorem 1.6 (Type preservation) *Se $t \to r$ e $\emptyset \vdash t{:}\,\tau$, allora r e' un valore v t.c. $\emptyset \vdash v{:}\,\tau$.*

Quindi la valutazione dei termini ben-tipati non puo' portare ad errori.

1.6 [Win93, Ch 11]: type system per HOFL con informazione di tipo

In molti linguaggi di programmazione fortemente tipati si specifica il tipo di un oggetto al momento della dichiarazione. Questo permette di semplificare notevolmente la ricerca di una derivazione nel type system. Consideriamo una variante di HOFL con informazione di tipo nei termini, e facciamo vedere come cio' si ripercuota a livello di type system. Riportiamo le produzioni che introducono informazione di tipo, le altre sono invariate:

$$t \in \mathsf{Exp}^* \quad ::= \quad \ldots \mid \lambda x{:}\tau.t \mid \mathsf{rec}\ x{:}\tau.t$$

Si noti che per il costrutto let-in non si e' aggiunto informazione di tipo. Si puo' definire nel modo ovvio una funzione $erase\colon \mathsf{Exp}^* \to \mathsf{Exp}$ di cancellazione della informazione di tipo.

Riportiamo le modifiche al type system in corrispondenza delle due produzioni modificate

- $\lambda \dfrac{\Gamma[x := \tau_1] \vdash t : \tau_2}{\Gamma \vdash \lambda x : \tau_1 . t : \tau_1 \to \tau_2}$ rec $\dfrac{\Gamma[x := \tau] \vdash t : \tau}{\Gamma \vdash \mathsf{rec}\ x : \tau . t : \tau}$

Per il type system modificato vale la proprieta' di univocita' del tipo (che non vale nel type system originario):

Proposition 1.7 (Univocita') *Se $\Gamma \vdash t : \tau_1$ e $\Gamma \vdash t : \tau_2$, allora $\tau_1 \equiv \tau_2$.*

La relazione tra i due sistema di tipo e' la seguente, dove indichiamo con $\Gamma \vdash_* t : \tau$ le asserzioni del type system per termini con informazione di tipo e $\Gamma \vdash t : \tau$ le asserzioni dell'altro type system.

Proposition 1.8 *Se $\Gamma \vdash_* t : \tau$, allora $\Gamma \vdash erase(t) : \tau$. Se $\Gamma \vdash t : \tau$, allora esiste $t' \in \mathsf{Exp}^*$ t.c. $\Gamma \vdash_* t' : \tau$ e $erase(t') \equiv t$.*

1.7 [Win93, Ch 11]: osservazioni ed equivalenza osservazionale per HOFL

Anzitutto dobbiamo fissare l'insieme P dei programmi e quello O delle osservazioni. Le scelte piu' ovvie per P sono:

- $t \in P_0 \overset{\Delta}{\iff} t$ e' un termine chiuso;
- $t \in P_{typed} \overset{\Delta}{\iff} t$ e' un termine chiuso e ben tipato, cioe' $\emptyset \vdash t : \tau$ per qualche tipo τ;
- $t \in P_{int} \overset{\Delta}{\iff} t$ e' un termine chiuso di tipo int, cioe' $\emptyset \vdash t : \mathsf{int}$.

Le scelte P_{typed} e P_{int} catturano l'idea che i programmi che non passano i controlli statici (in questo caso il type-checking) non vengono compilati, e quindi non sono osservabili a tempo di esecuzione. Indipendentemente da quale P si sceglie, come osservazioni le scelte piu' naturali sono:

- "t termina" $\overset{\Delta}{\iff} \exists v . t \to v$;
- "t ha valore n" $\overset{\Delta}{\iff} t \to n$, particolarmente appropriata per programmi $t \in P_{int}$.

Se consideriamo solo programmi tutti i termini chiusi, ci sono dei problemi tecnici ad avere un solo tipo di frammenti, oveero Exp, perche' un contesto $C[\square]$ puo' legare solo un numero finito di variabili. Fissato $X \subseteq_{fin} \mathsf{Var}$ possiamo definire un insieme di frammenti $t \in E_X \overset{\Delta}{\iff} \mathrm{FV}(t) \subseteq X$, ed un corrispondente insieme di contesti (con un buco) $C[\square] \in C_X \overset{\Delta}{\iff} \forall t \in E_X . C[t] \in P_0$. In questo modo, abbiamo una equivalenza $\approx_X$ su E_X, che scriveremo $X \vdash t_1 \approx t_2$. Se consideriamo solo programmi ben tipati, p.e. prendiamo P_{int}, e' naturale imporre lo stesso requisito sui frammenti di programma. Fissati Γ e τ possiamo definire un insieme di frammenti $t \in E_{\Gamma,\tau} \overset{\Delta}{\iff} \Gamma \vdash t : \tau$, ed un insieme di contesti (con un buco) $C[\square] \in C_{\Gamma,\tau} \overset{\Delta}{\iff} \forall t \in E_{\Gamma,\tau} . C[t] \in P_{int}$. In questo modo, abbiamo una equivalenza $\approx_{\Gamma,\tau}$ su $E_{\Gamma,\tau}$ per ogni Γ e τ, che scriveremo $\Gamma \vdash t_1 \approx t_2 : \tau$.

1.8 Esercizi

IMP con exit e continue. Estendere i comandi di IMP con

- exit, che fa' uscire dal ciclo while corrente, e passa al prossimo comando dopo il ciclo while, quindi $(\mathsf{while\ true\ do\ exit}), s \to s$
- continue, che fa' uscire dal dal ciclo while corrente, e torna ad eseguere il comando while, quindi il seguente comando non termina $x := 0; (\mathsf{while}\ x = 0\ \mathsf{do}\ (\mathsf{continue}; x := 1))$

1) Modificare la semantica operazionale a grandi passi per i comandi di IMP [NOTA occorre rivedere le asserzioni $c, s \to s'$ per l'esecuzione dei comandi, e quindi oltre ad aggiungere delle regole per continue ed exit occorre anche rivedere le regole per gli altri comandi]

2) Modificare l'analisi statica per IMP, in modo da controllare anche se si fa' un uso appropriato di continue ed exit, ovvero se tali comandi sono usati solo nel corpo di cicli while. Enunciare la proprieta' che asserisce la "correttezza" dell'analisi statica rispetto alla semantica operazione

HOFL con or. Estendere la BNF per i termini di HOFL con la produzione $t \in \mathsf{Exp} ::= \ldots \mid t_1\ \mathsf{or}\ t_2$ [NOTA ci si puo' restringere al frammento di HOFL senza let-in e senza i costrutti per manipolare coppie]. Il costrutto $t_1\ \mathsf{or}\ t_2$ sceglie in modo non-deterministico se valutare t_1 o t_2.

1) Estendere la semantica operazione CBV $t \to v$ con delle regole per la valutazione di $t_1\ \mathsf{or}\ t_2$, senza modificare le asserzioni $t \to v$ [NOTA in questo contesto l'asserzione $t \to v$ significa che v e' uno dei possibili valori di t].

2) considerare una nuova semantica operazione CBV per HOFL con or, dove le asserzioni sono $t \to V$ e V e' un insieme finito di valori, ed il significato dell'asserzione e' V e' l'insieme dei possibili valori di t [e tutte le possibili computazioni di t terminano]. Quindi si avra' $1\ \mathsf{or}\ 2 \to \{1, 2\}$, mentre $1\ \mathsf{or}\ (\mathsf{rec}\ x.x) \to V$ non e' derivabile per nessuna scelta di V, perche' $(\mathsf{rec}\ x.x)$ non termina. Dire se vale la proprieta' di univocita' $t \to V_1 \wedge t \to V_2 \Longrightarrow V_1 = V_2$.

3) Usando le semantiche operazionali dei punti 1) e 2) definire le osservazioni "t puo' terminare" [cioe' almeno una computazione di t termina] e "t deve terminare" [cioe' tutte le computazioni di t terminano]. In particolare, se t e' il termine $1\ \mathsf{or}\ (\mathsf{rec}\ x.x)$ si avra' che "t puo' terminare", ma non e' vero che "t deve terminare".

4) definire una semantica operazionale CBV a piccoli passi per HOFL con or, con asserzioni della forma $t \to t'$. Dire come definire la semantica delle osservazioni al punto 3) in termini di questa semantica a piccoli passi.

Combinazione di HOFL e IMP. Si consideri la seguente estensione di HOFL, caratterizzata dall'aggiunta delle locazioni (contenenti interi, come in IMP):

$$\begin{array}{rcl}
l \in \mathsf{Loc} & ::= & \text{locazioni} \\
n \in \mathsf{N} & ::= & \text{i numeri interi} \\
x \in \mathsf{Var} & ::= & \ldots \\
t \in \mathsf{Exp} & ::= & n \mid t_1 + t_2 \mid \mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \mid x \mid \lambda x.t \mid t_1\ t_2 \mid \mathsf{rec}\ x.t \mid l \mid t_1 := t_2 \mid\ !t \\
v \in \mathsf{Val} & ::= & n \mid \lambda x.t \mid l \\
\tau \in \mathsf{Type} & ::= & \mathsf{int} \mid \mathsf{loc} \mid \tau_1 \to \tau_2
\end{array}$$

dove i nuovi costrutti hanno la seguente semantica informale

- $t_1 := t_2$, se t_1 e' una locazione l e t_2 ha valore n, allora assegna n ad l e restituisci l
- $!t$, se t e' una locazione l, allora restituisci il contenuto n di l

1) Estendere la definizione (data per HOFL) dell'insieme $FV(t) \subseteq_{fin} \mathsf{Var}$ delle variabili libere. Si consiglia di usare una tabella

$t \in \mathsf{Exp}$	$FV(t)$
n	$\emptyset$
...	...

2) Estendere il type system (dato per HOFL) per derivare asserzioni della forma $\Gamma \vdash t : \tau$.

3) Dare una semantica operazionale a grandi passi. Prima fissare la forma delle asserzioni (e gli eventuali domini semantici, p.e. quello S degli stati), poi dare le regole.

4) Formulare un analogo del teorema di Type preservation dato per HOFL, spiegare che tipo di induzione va' usata per dimostrarlo, ed illustrare i casi piu' significativi della dimostrazione, ovvero: $t_1 + t_2$, $t_1\ t_2$, $t_1 := t_2$ e $!t$.

HOFL con eccezioni. Si consideri la seguente variante di HOFL che aggiunge dei costrutti tipo CAML per la gestione delle eccezioni

$$
\begin{array}{rcl}
exn \in \mathsf{Exn} & ::= & \text{nomi di eccezione, p.e. i numeri naturali} \\
n \in \mathsf{N} & ::= & \text{i numeri interi} \\
x \in \mathsf{Var} & ::= & \ldots \\
t \in \mathsf{Exp} & ::= & n \mid t_1 + t_2 \mid \mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2 \mid x \mid \lambda x.t \mid t_1\ t_2 \mid \mathsf{rec}\ x.t \mid \\
 & & exn \mid \mathsf{raise}(t) \mid \mathsf{newexn}\ x\ \mathsf{in}\ t \mid t_0\ \mathsf{handle}\ t_1\ \mathsf{with}\ t_2 \\
v \in \mathsf{Val} & ::= & n \mid \lambda x.t \mid exn \\
\tau \in \mathsf{Type} & ::= & \mathsf{int} \mid \mathsf{exc} \mid \tau_1 \to \tau_2
\end{array}
$$

dove i nuovi costrutti hanno la seguente semantica informale

- $\mathsf{raise}(t)$, se t si valuta in una eccezione exn (gia' esistente), allora solleva l'eccezione exn
- $\mathsf{newexn}\ x\ \mathsf{in}\ t$, rimpiazza x (in t) con una nuova eccezione exn, e poi valuta t
- $t_0\ \mathsf{handle}\ t_1\ \mathsf{with}\ t_2$, se t_0 non solleva eccezioni, si comporta come t_0; se invece t_0 solleva una eccezione exn (gia' esistente) e t_1 vi valuta in exn, allora si valuta t_2

1. Definire l'insieme $FV(t) \subseteq_{fin} \mathsf{Var}$ delle variabili libere e $UE(t) \subseteq_{fin} \mathsf{Exn}$ delle eccezioni usate in t. **Per evitare confusione tra interi e eccezioni, scriveremo exn_n per indicare la eccezione corrispondente al numero naturale n.** Usare una tabella

$t \in \mathsf{Exp}$	$FV(t)$	$UE(t)$
exn	$\emptyset$	$\{exn\}$
$\ldots$	$\ldots$	$\ldots$

2. Dare una semantica operazionale a grandi passi, con asserzioni della forma $t, \mu \to \mathsf{val}\ v, \mu'$ e $t, \mu \to \mathsf{exc}\ exn, \mu'$ dove $t \in \mathsf{Exp}_0$, $v \in \mathsf{Val}_0$ e $\mu, \mu' \in \mathsf{Nat}$ sono contatori delle eccezioni. In altre parole μ indica che sono state generate μ eccezioni $exn \in \mu = \{0, 1, \ldots, \mu - 1\}$ (cioe' identifichiamo un numero naturale con l'insieme dei suoi predecessori). Per esempio, le seguenti asserzioni devono essere derivabili:

   $exn, \mu \to \mathsf{val}\ exn, \mu \quad \mathsf{raise}(exn), \mu \to \mathsf{exc}\ exn, \mu \quad \mathsf{newexn}\ x\ \mathsf{in}\ x, \mu \to \mathsf{val}\ exn_\mu, \mu + 1$

3. La semantica operazionale deve formalizzare la gestione dei nomi di eccezione stile CAML.

   Dimostrare che $t, \mu \to r, \mu' \Longrightarrow \mu \le \mu'$, cioe' il numero di nomi di eccezione cresce come effetto della valutazione (questo a causa del costrutto newexc-in).

4. Dimostrare che

   $t, \mu \to \mathsf{exc}\ exn, \mu' \wedge UE(t) \subseteq \mu \Longrightarrow exn \in \mu'$
   $t, \mu \to \mathsf{val}\ v, \mu' \wedge UE(t) \subseteq \mu \Longrightarrow UE(v) \subseteq \mu'$

5. Estendere il type system (dato per HOFL) per derivare asserzioni della forma $\Gamma \vdash t : \tau$.

6. Formulare un analogo del teorema di Type preservation dato per HOFL, spiegare che tipo di induzione va' usata per dimostrarlo, ed illustrare i casi piu' significativi della dimostrazione.

   ATTENZIONE: occorre tenere conto che ci sono due asserzioni della semantica operazionale $t, \mu \to r, \mu'$ e che non basta assumere $\emptyset \vdash t : \tau$, ci vuole anche $UE(t) \subseteq \mu$.

   $t, \mu \to \mathsf{val}\ v, \mu' \wedge UE(t) \subseteq \mu \wedge \emptyset \vdash t : \tau \Longrightarrow \emptyset \vdash v : \tau$

Soluzione.

1.

$t \in \mathsf{Exp}$	$FV(t)$	$UE(t)$
n	$\emptyset$	$\emptyset$
$t_1 + t_2$	$FV(t_1) \cup FV(t_2)$	$UE(t_1) \cup UE(t_2)$
$\mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2$	$FV(t_0) \cup FV(t_1) \cup FV(t_2)$	$UE(t_0) \cup UE(t_1) \cup UE(t_2)$
x	$\{x\}$	$\emptyset$
$\lambda x.t$	$FV(t) - \{x\}$	$UE(t)$
$t_1\ t_2$	$FV(t_1) \cup FV(t_2)$	$UE(t_1) \cup UE(t_2)$
$\mathsf{rec}\ x.t$	$FV(t) - \{x\}$	$UE(t)$
exn	$\emptyset$	$\{exn\}$
$\mathsf{raise}(t)$	$FV(t)$	$UE(t)$
$\mathsf{newexn}\ x\ \mathsf{in}\ t$	$FV(t) - \{x\}$	$UE(t)$
$t_0\ \mathsf{handle}\ t_1\ \mathsf{with}\ t_2$	$FV(t_0) \cup FV(t_1) \cup FV(t_2)$	$UE(t_0) \cup UE(t_1) \cup UE(t_2)$

2. $$\frac{}{n,\mu_0 \to \mathsf{val}\ n,\mu_0} \qquad \frac{t_1,\mu_0 \to \mathsf{val}\ n_1,\mu_1 \qquad t_2,\mu_1 \to \mathsf{val}\ n_2,\mu_2}{t_1+t_2,\mu_0 \to \mathsf{val}\ n,\mu_2}\ n=n_1+n_2$$

$$\frac{t_1,\mu_0 \to \mathsf{exc}\ exn,\mu_1}{t_1+t_2,\mu_0 \to \mathsf{exc}\ exn,\mu_1} \qquad \frac{t_1,\mu_0 \to \mathsf{val}\ n_1,\mu_1 \qquad t_2,\mu_1 \to \mathsf{exc}\ exn,\mu_2}{t_1+t_2,\mu_0 \to \mathsf{exc}\ exn,\mu_2}$$

$$\frac{t_0,\mu_0 \to \mathsf{val}\ 0,\mu_1 \qquad t_1,\mu_1 \to r,\mu_2}{\mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2,\mu_0 \to r,\mu_2} \qquad \frac{t_0,\mu_0 \to \mathsf{val}\ n,\mu_1 \qquad t_2,\mu_1 \to r,\mu_2}{\mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2,\mu_0 \to r,\mu_2}\ n\neq 0$$

$$\frac{t_0,\mu_0 \to \mathsf{exc}\ exn,\mu_1}{\mathsf{if}\ t_0\ \mathsf{then}\ t_1\ \mathsf{else}\ t_2,\mu_0 \to \mathsf{exc}\ exn,\mu_1} \qquad \frac{}{\lambda x.t,\mu_0 \to \mathsf{val}\ \lambda x.t,\mu_0} \qquad \frac{\begin{array}{l} t_1,\mu_0 \to \mathsf{val}\ \lambda x.t,\mu_1 \\ t_2,\mu_1 \to \mathsf{val}\ v,\mu_2 \\ t[x{:=}\,v],\mu_2 \to r',\mu_3 \end{array}}{t_1\ t_2,\mu_0 \to r',\mu_3}$$

$$\frac{t_1,\mu_0 \to \mathsf{exc}\ exn,\mu_1}{t_1\ t_2,\mu_0 \to \mathsf{exc}\ exn,\mu_1} \qquad \frac{\begin{array}{l} t_1,\mu_0 \to \mathsf{val}\ \lambda x.t,\mu_1 \\ t_2,\mu_1 \to \mathsf{exc}\ exn,\mu_2 \end{array}}{t_1\ t_2,\mu_0 \to \mathsf{exc}\ exn,\mu_2} \qquad \frac{t[x{:=}\,\mathsf{rec}\ x.t],\mu_0 \to r,\mu_1}{\mathsf{rec}\ x.t,\mu_0 \to r,\mu_1}$$

$$\frac{}{exn,\mu_0 \to \mathsf{val}\ exn,\mu_0} \qquad \frac{t,\mu_0 \to \mathsf{val}\ exn,\mu_1}{\mathsf{raise}(t),\mu_0 \to \mathsf{exc}\ exn,\mu_1} \qquad \frac{t,\mu_0 \to \mathsf{exc}\ exn,\mu_1}{\mathsf{raise}(t),\mu_0 \to \mathsf{exc}\ exn,\mu_1}$$

$$\frac{t[x{:=}\,exn_{\mu_0}],\mu_0+1 \to r,\mu_1}{\mathsf{newexn}\ x\ \mathsf{in}\ t,\mu_0 \to r,\mu_1} \qquad \frac{t_0,\mu_0 \to \mathsf{val}\ v,\mu_1}{t_0\ \mathsf{handle}\ t_1\ \mathsf{with}\ t_2,\mu_0 \to \mathsf{val}\ v,\mu_1}$$

$$\frac{t_0,\mu_0 \to \mathsf{exc}\ exn,\mu_1 \qquad t_1,\mu_1 \to \mathsf{val}\ exn,\mu_2 \qquad t_2,\mu_2 \to r,\mu_3}{t_0\ \mathsf{handle}\ t_1\ \mathsf{with}\ t_2,\mu_0 \to r,\mu_3}$$

$$\frac{\begin{array}{c} t_0,\mu_0 \to \mathsf{exc}\ exn,\mu_1 \\ t_1,\mu_1 \to \mathsf{val}\ exn',\mu_2 \end{array}}{t_0\ \mathsf{handle}\ t_1\ \mathsf{with}\ t_2,\mu_0 \to \mathsf{exc}\ exn,\mu_2}\ exn' \not\equiv exn \qquad \frac{\begin{array}{c} t_0,\mu_0 \to \mathsf{exc}\ exn,\mu_1 \\ t_1,\mu_1 \to \mathsf{exc}\ exn',\mu_2 \end{array}}{t_0\ \mathsf{handle}\ t_1\ \mathsf{with}\ t_2,\mu_0 \to \mathsf{exc}\ exn,\mu_2}$$

3. $t,\mu \to r,\mu' \Longrightarrow \mu \leq \mu'$ si dimostra per induzione sulla derivazione di $t,\mu \to r,\mu'$

4. $t,\mu \to \mathsf{exc}\ exn,\mu' \wedge UE(t) \subseteq \mu \Longrightarrow exn \in \mu'$
   $t,\mu \to \mathsf{val}\ v,\mu' \wedge UE(t) \subseteq \mu \Longrightarrow UE(v) \subseteq \mu'$

   si dimostrano per induzione sulla derivazione di $t,\mu \to r,\mu'$. Tuttavia nelle regole dove si effettuano delle sostituzioni, si deve sfruttare anche la seguente **proprieta' sostitutiva di UE** $UE(t_2[x{:=}\,t_1]) \subseteq UE(t_1) \cup UE(t_1)$.

2 Lambda-calcolo e categorie cartesianamente chiuse

In questa nota si definisce la sintassi del λ-calcolo semplice, che useremo come *metalinguaggio* per la semantica denotazionale, e la sua interpretazione in una categoria cartesianamente chiusa (CCC).

2.1 Sintassi

Le categorie sintattiche sono

$$\begin{array}{rcl}
C \in \mathsf{TCon} & ::= & \text{costruttori di tipo} \\
X \in \mathsf{TVar} & ::= & \text{variabili di tipo} \\
\tau \in \mathsf{Type} & ::= & X \mid C\,\overline{\tau} \\
 & \mid & \tau_1 \to \tau_2 \text{ esponenziale} \\
 & \mid & (\tau_i|i \in n) \text{ prodotto finito} \\
 & \mid & [\tau_i|i \in n] \text{ somma finita} \\
\sigma \in \mathsf{TSchema} & ::= & \tau \mid \forall X.\sigma \\
c \in \mathsf{Con} & ::= & \text{costanti polimorfe} \\
x \in \mathsf{Var} & ::= & \text{variabili} \\
e \in \mathsf{Exp} & ::= & x \mid c[\overline{\tau}] \\
 & \mid & \lambda x{:}\tau.e \ \lambda\text{-astrazione} \\
 & \mid & e_1\ e_2 \text{ applicazione} \\
 & \mid & (e_i|i \in n)\ n\text{-upla} \\
 & \mid & \pi_i(e) \text{ proiezione} \\
 & \mid & in_i(e) \text{ inclusione} \\
 & \mid & \mathsf{case}\ e\ \mathsf{of}\ [in_i(x_i) \Rightarrow e_i|i \in n] \\
\Gamma \in \mathsf{Ctx} & ::= & \emptyset \mid \Gamma, x{:}\tau \\
\Sigma \in \mathsf{Sig} & ::= & \emptyset \mid \Sigma, C{:}n \mid \Sigma, c{:}\sigma
\end{array}$$

I contesti Γ sono sequenze di dichiarazioni $x{:}\tau$, una variabile puo' essere dichiarata piu' volte in Γ, in tal caso la dichiarazione piu' a destra ha la precedenza sulle altre. Questa definizione dei contesti e' la piu' conveniente per definire l'interpretazione dei termini in un CCC.

Il motivo principale introdurre le variabili di tipo X e' la definizione degli schemi di tipo σ, che servono per tipare le costanti polimorfe. A livello del type system faremo l'ipotesi semplificativa che tutti i tipi τ usati nelle asserzioni $\Gamma \vdash e{:}\tau$ siano chiusi.

Una segnatura Σ per il λ-calcolo e' simile ad una segnatura algebrica, ed essa permette di adattare il linguaggio per meglio riflettere le peculiarita' di una particolare CCC.

2.2 Regole di buona formazione

Diamo delle regole di buona formazione per segnature, tipi e contesti. Le asserzioni che consideriamo sono

- $\Sigma \vdash$, cioe' Σ e' una segnatura ben formata
- $\Sigma; \overline{X} \vdash \sigma$, cioe' σ e' uno schema di tipo ben formato con varibili di tipo in $\overline{X}$ (sequenza di variabili di tipo)
- $\Sigma \vdash \Gamma$, cioe' Γ e' uno contesto ben formato

$$\Sigma.\emptyset\ \frac{}{\emptyset \vdash} \qquad \Sigma.C\ \frac{\Sigma \vdash}{\Sigma, C{:}n \vdash}\ C \text{ new} \qquad \Sigma.c\ \frac{\Sigma;\emptyset \vdash \sigma}{\Sigma, c{:}\sigma \vdash}\ c \text{ new} \qquad \Gamma.\emptyset\ \frac{\Sigma \vdash}{\Sigma \vdash \emptyset} \qquad \Gamma.x\ \frac{\Sigma \vdash \Gamma \qquad \Sigma;\emptyset \vdash \tau}{\Sigma \vdash \Gamma, x{:}\tau}$$

$$X\ \frac{\Sigma \vdash}{\Sigma;\overline{X} \vdash X}\ X \text{ in } \overline{X} \qquad C\ \frac{\{\Sigma;\overline{X} \vdash \tau_i \mid i \in n\}}{\Sigma;\overline{X} \vdash C\ \tau_{i \in n}}\ \Sigma(C) = n \qquad \forall\ \frac{\Sigma;\overline{X}, X \vdash \sigma}{\Sigma;\overline{X} \vdash \forall X.\sigma}$$

$$\to\ \frac{\Sigma;\overline{X} \vdash \tau_1 \qquad \Sigma;\overline{X} \vdash \tau_2}{\Sigma;\overline{X} \vdash \tau_1 \to \tau_2} \qquad \times\ \frac{\{\Sigma;\overline{X} \vdash \tau_i \mid i \in n\}}{\Sigma;\overline{X} \vdash (\tau_i|i \in n)} \qquad +\ \frac{\{\Sigma;\overline{X} \vdash \tau_i \mid i \in n\}}{\Sigma;\overline{X} \vdash [\tau_i|i \in n]}$$

Proposition 2.1 *L'insieme delle regole di buona formazione e' deterministico.*

2.3 Type system

Il type system che definisce i termini ben tipati $\Gamma \vdash e\colon\tau$ e' parametrizzato rispetto ad una segnatura ben formata Σ. Quando vogliamo esplicitare a quale Σ ci riferiamo, scriveremo $\Gamma \vdash_\Sigma e\colon\tau$. Nelle side-conditions del type system si fa uso *implicito* di asserzioni di buona formazione, al fine di garantire che $\Sigma \vdash \Gamma$ e $\Sigma;\emptyset \vdash \tau$.

$$x\ \frac{}{\Gamma, x\colon\tau \vdash x\colon\tau} \qquad x'\ \frac{\Gamma \vdash x\colon\tau}{\Gamma, x'\colon\tau' \vdash x\colon\tau} \qquad c\ \frac{}{\Gamma \vdash c[\overline{\tau}]\colon\tau[\overline{X} := \overline{\tau}]}\ \Sigma(c) \equiv \forall\overline{X}.\tau \wedge |\overline{X}| = |\overline{\tau}| \wedge \Sigma;\emptyset \vdash \overline{\tau}$$

$$\to.\mathrm{I}\ \frac{\Gamma, x\colon\tau_1 \vdash e\colon\tau_2}{\Gamma \vdash \lambda x\colon\tau_1.e\colon\tau_1 \to \tau_2} \qquad \to.\mathrm{E}\ \frac{\Gamma \vdash e_1\colon\tau_1 \to \tau_2 \qquad \Gamma \vdash e_2\colon\tau_1}{\Gamma \vdash e_1\ e_2\colon\tau_2}$$

$$\times.\mathrm{I}\ \frac{\{\Gamma \vdash e_i\colon\tau_i | i \in n\}}{\Gamma \vdash (e_i | i \in n)\colon(\tau_i | i \in n)} \qquad \times.\mathrm{E}\ \frac{\Gamma \vdash e\colon(\tau_i | i \in n)}{\Gamma \vdash \pi_i(e)\colon\tau_i}\ i \in n$$

$$+.\mathrm{I}\ \frac{\Gamma \vdash e\colon\tau_i}{\Gamma \vdash in_i(e)\colon[\tau_i | i \in n]}\ i \in n \qquad +.\mathrm{E}\ \frac{\Gamma \vdash e\colon[\tau_i | i \in n] \qquad \{\Gamma, x_i\colon\tau_i \vdash e_i\colon\tau | i \in n\}}{\Gamma \vdash \mathsf{case}\ e\ \mathsf{of}\ [in_i(x_i) \Rightarrow e_i | i \in n]\colon\tau}$$

Proposition 2.2 (Univocita') *Se si escludono i tipi somma, allora la derivazione di un termine ben tipato $\Gamma \vdash e\colon\tau$ e' unica, e τ e' univocamente determinato da Γ ed e.*

Remark 2.3 Il modo piu' semplice per forzare il risultato di univocita' anche in presenza di tipi somma e' di aggiungere piu' informazione di tipo nei termini, ovvero modificando la regola $+.\mathrm{I}\ \dfrac{\Gamma \vdash e\colon\tau_i}{\Gamma \vdash in_i[\tau_{i\in n}](e)\colon[\tau_i | i \in n]}\ i \in n$.

2.4 Interpretazione in una CCC (con somme)

Nel definire l'interpretazione si puo' fare l'ipotesi semplificativa che la segnatura Σ contenga solo costanti monomorfe $c\colon\tau_c$, purche' si permetta di avere di avere segnature con un numero infinito di simboli. Per esempio, una costante polimorfa $c\colon\forall X.\tau_c$ si puo' sostituire con una infinita' di costanti monomorfe $c_\tau\colon\tau_c[X := \tau]$, una per ogni tipo chiuso τ. Fissata una CCC $\mathcal{C}$ con una scelta prefissata di prodotti finiti, esponenziali (e somme), l'interpretazione mappa un tipo τ ben formato, cioe' $\Sigma;\emptyset \vdash \tau$ e' derivabile, in un oggetto $[\![\tau]\!]$ di $\mathcal{C}$, ed un termine ben tipato $\Gamma \vdash e\colon\tau$ in un morfismo di $\mathcal{C}$ da $[\![\Gamma]\!]$ a $[\![\tau]\!]$. L'interpretazione dipende dalla interpretazione dei simboli nella segnatura Σ:

- un costruttore di tipo $C\colon n$ n-ario e' interpretato da una funzione $[\![C]\!]\colon\mathcal{C}_0^n \to \mathcal{C}_0$
- una costante monomorfa $c\colon\tau$ e' interpretata da un morfismo $[\![c]\!]\colon 1 \to [\![\tau]\!]$.

L'interpretazione e' definita per induzione sulla struttura di τ e Γ:

- interpretazione dei tipi chiusi

$$[\![C\ \tau_{i\in n}]\!] \stackrel{\Delta}{=} [\![C]\!]([\![\tau_i]\!] | i \in n) \qquad [\![\tau_1 \to \tau_2]\!] \stackrel{\Delta}{=} [\![\tau_2]\!]^{[\![\tau_1]\!]} \qquad [\![(\tau_i | i \in n)]\!] \stackrel{\Delta}{=} \prod_{i\in n}[\![\tau_i]\!] \qquad [\![[\tau_i | i \in n]]\!] \stackrel{\Delta}{=} \coprod_{i\in n}[\![\tau_i]\!]$$

- interpretazione dei contesti $\quad [\![\emptyset]\!] \stackrel{\Delta}{=} 1$, cioe' l'oggetto terminale $\quad [\![\Gamma, x\colon\tau]\!] \stackrel{\Delta}{=} [\![\Gamma]\!] \times [\![\tau]\!]$

Per definire l'interpretazione dei termini ben tipati (e piu' precisamente del costrutto case-of) si sfrutta la proprieta' distribuitiva del prodotto rispetto alla somma.

Proposition 2.4 *In una CCC il morfismo* canonico $\coprod_{i\in n} a \times b_i \dashrightarrow a \times \coprod_{i\in n} b_i$ *e' un isomorfismo.*

$$\begin{tikzcd}
\coprod_{i\in n} a \times b_i \arrow[rr, dashed] & & a \times \coprod_{i\in n} b_i \\
a \times b_i \arrow[u, "\iota_i"] \arrow[urr, "\mathrm{id}_a \times \iota_i"'] & &
\end{tikzcd}$$

L'interpretazione di un termine ben tipato $\Gamma \vdash e\colon\tau$ e' data per induzione sulla derivazione di $\Gamma \vdash e\colon\tau$. Per convenienza, diamo un nome agli oggetti che interpretano i seguenti contesti/tipi $a = [\![\Gamma]\!]$, $b = [\![\tau]\!]$, $b' = [\![\tau']\!]$ e $b_i = [\![\tau_i]\!]$.

$$\frac{}{[\![\Gamma, x{:}\,\tau \vdash x{:}\,\tau]\!] \triangleq a \times b \xrightarrow{\pi_2} b}$$

$$\frac{[\![\Gamma \vdash x{:}\,\tau]\!] = a \xrightarrow{f} b}{[\![\Gamma, x'{:}\,\tau' \vdash x{:}\,\tau]\!] \triangleq a \times b' \xrightarrow{\pi_1} a \xrightarrow{f} b}$$

$$\frac{}{[\![\Gamma \vdash c{:}\,\tau]\!] \triangleq a \xrightarrow{!} 1 \xrightarrow{[\![c]\!]} a}$$

$$\frac{[\![\Gamma, x{:}\,\tau_1 \vdash e{:}\,\tau_2]\!] = a \times b_1 \xrightarrow{f} b_2}{[\![\Gamma \vdash \lambda x{:}\,\tau_1.e{:}\,\tau_1 \to \tau_2]\!] \triangleq a \xrightarrow{\Lambda f} b_2^{b_1}}$$

$$\frac{[\![\Gamma \vdash e_1{:}\,\tau_1 \to \tau_2]\!] = a \xrightarrow{f_1} b_2^{b_1} \qquad [\![\Gamma \vdash e_2{:}\,\tau_1]\!] = a \xrightarrow{f_2} b_1}{[\![\Gamma \vdash e_1\ e_2{:}\,\tau_2]\!] \triangleq a \xrightarrow{\langle f_1, f_2\rangle} b_2^{b_1} \times b_1 \xrightarrow{ev} b_2}$$

$$\frac{[\![\{\Gamma \vdash e_i{:}\,\tau_i | i \in n\}]\!] = a \xrightarrow{f_i} b_i \quad i \in n}{[\![\Gamma \vdash (e_i | i \in n){:}\,(\tau_i | i \in n)]\!] \triangleq a \xrightarrow{\langle f_i | i \in n\rangle} \prod_{i\in n} b_i}$$

$$\frac{[\![\Gamma \vdash e{:}\,(\tau_i | i \in n)]\!] = a \xrightarrow{f} \prod_{i\in n} b_i}{[\![\Gamma \vdash \pi_i(e){:}\,\tau_i]\!] \triangleq a \xrightarrow{f} \prod_{i\in n} b_i \xrightarrow{\pi_i} b_i}$$

$$\frac{[\![\Gamma \vdash e{:}\,\tau_i]\!] = a \xrightarrow{f_i} b_i}{[\![\Gamma \vdash in_i(e){:}\,[\tau_i | i \in n]]\!] \triangleq a \xrightarrow{f_i} b_i \xrightarrow{\iota_i} \coprod_{i\in n} b_i}$$

$$\frac{[\![\Gamma \vdash e{:}\,[\tau_i | i \in n]]\!] = a \xrightarrow{f} \coprod_{i\in n} b_i \qquad [\![\Gamma, x_i{:}\,\tau_i \vdash e_i{:}\,\tau]\!] = a \times b_i \xrightarrow{f_i} b \quad i \in n}{[\![\Gamma \vdash \mathsf{case}\ e\ \mathsf{of}\ [in_i(x_i) \Rightarrow e_i | i \in n]{:}\,\tau]\!] \triangleq a \xrightarrow{\langle \mathrm{id}_a, f\rangle} a \times \coprod_{i\in n} b_i \cong \coprod_{i\in n} a \times b_i \xrightarrow{[f_i | i \in n]} b}$$

2.5 Metalinguaggio e semantica denotazionale

Per semantica denotazionale di un linguaggio (di programmazione) PL in una CCC $\mathcal{C}$, si intende una interpretazione della sintassi di PL usando oggetti e morfismi di $\mathcal{C}$. Avendo a disposizione un metalinguaggio ML per CCC, questa interpretazione puo' essere data in modo *indiretto* mediante una traduzione da PL a ML. Questo presenta notevoli vantaggi:

- gli strumenti matematici necessari per definire una traduzione sono molto piu' elementari di quelli necessari per dare una interpretazione
- le peculiarita' della CCC $\mathcal{C}$ possono essere in gran parte ignorate, o piu' precisamente incorporate nel metalinguaggio fissando una segnatura Σ (scriveremo $ML(\Sigma)$ per il metalinguaggio ML con la segnatura Σ fissata)
- una traduzione da PL a $ML(\Sigma)$ descrive in effetti tutta una classe di interpretazioni di PL, che si ottengono cambiando la CCC $\mathcal{C}$ e/o l'interpretazione in $\mathcal{C}$ dei simboli della segnatura Σ
- un metalinguaggio $ML(\Sigma)$ (ed una sua interpretazione in una CCC $\mathcal{C}$) puo' essere utilizzato come target per tradurre piu' linguaggi (di programmazione) PL, semplicemente cambiando la funzione di traduzione.

$$\begin{tikzcd}
PL_1 \arrow[ddr] & \dots & PL_m \arrow[ddl] & \text{Linguaggi di Programmazione} \\
& & & \text{traduzioni} \\
& ML(\Sigma) \arrow[ddl] \arrow[ddr] & & \text{MetaLinguaggio con segnatura } \Sigma \text{ fissata} \\
& & & \text{interpretazioni, univocamente determinate dalla interpretazione di } \Sigma \\
\mathcal{C}_1 & \dots & \mathcal{C}_n & \text{Categorie Cartesianamente Chiuse}
\end{tikzcd}$$

Come dare la semantica denotazione. Fissato il linguaggio PL e la CCC $\mathcal{C}$ in cui lo si vuole interpretare, per specificare completamente una interpretazione di PL in $\mathcal{C}$ (tramite ML) occorrera':

- fissare una segnatura Σ per ML
- dare una traduzione di PL in $ML(\Sigma)$, in genere definita per induzione
- dare una interpretazione di Σ in $\mathcal{C}$, in genere cio' richiede di sfruttare proprieta' specifiche della categoria $\mathcal{C}$.

Per i nostri scopi ci limiteremo a considerare la categoria **Set** degli insiemi e la categoria **Cpo** dei cpo.

Confronto tra semantica operazionale e denotazionale. La semantica operazionale e' essenzialmente la descrizione di un *interprete* per il linguaggio PL. La semantica denotazionale, una volta fissata la segnatura Σ e la sua interpretazione in $\mathcal{C}$, e' essenzialmente una *compilatore* da PL a $ML(\Sigma)$.

Esempi di segnature. Diamo alcuni esempi di segnatura per ML e loro possibili interpretazioni.

- Nat: 0 tipo,
  $z{:}\,Nat$ zero, $s{:}\,Nat \to Nat$ successore
  $I{:}\,\forall X.X \to (X \to X) \to Nat \to X$ inizialita'

  tale segnatura si interpreta in una CCC con natual number object
  $$\begin{tikzcd}
  1 \arrow[r, "z"] \arrow[dr, "a"'] & N \arrow[r, "s"] \arrow[d, dashed, "h"] & N \arrow[d, dashed, "h"] \\
  & X \arrow[r, "f"'] & X
  \end{tikzcd}$$

- $List$: 1 costruttore di tipo unario,
  $nil{:}\,\forall X.List\ X$, $cons{:}\,\forall X.X \to List\ X \to List\ X$,
  $I{:}\,\forall X,Y.X \to (X \to Y \to Y) \to List\ X \to Y$ inizialita'

  tale segnatura si interpreta in una CCC con list objects
  $$\begin{tikzcd}
  1 \arrow[r, "nil"] \arrow[dr, "a"'] & LX \arrow[d, dashed, "h"] & X \times LX \arrow[l, "cons"'] \arrow[d, dashed, "\mathrm{id}_X \times h"] \\
  & Y & X \times Y \arrow[l, "f"]
  \end{tikzcd}$$

- $fix{:}\,\forall X.(X \to X) \to X$

  tale costante si interpreta nella categoria dei cpo con bottom mediante l'operatore di minimo punto fisso.

- L: 1 costruttore di tipo unario,
  $val{:}\,\forall X.X \to L\ X$, $let{:}\,\forall X,Y.(X \to L\ Y) \to L\ X \to L\ Y$
  $\bot{:}\,\forall X.L\ X$, $fix{:}\,\forall X.(L\ X \to L\ X) \to L\ X$

  tale segnatura si interpreta nella categoria dei cpo definendo $[\![L]\!](\underset{\sim}{D}) \triangleq \underset{\sim}{D}_\bot$.

Anche i tipi prodotto e somma potevano essere introdotti mediante una segnatura.

3 Tipi computazionali e semantica denotazionale

Tra le varie segnature che si possono considerare per il metalinguaggio ML per CCC, vi e'

- la segnatura Σ_M per i **tipi computazionali**
  M: 1 costruttore di tipo unario,
  $val_M : \forall X.X \to M\ X$,
  $let_M : \forall X, Y.(X \to M\ Y) \to M\ X \to M\ Y$
- che puo' essere ulteriormente estesa con operazioni polimorfe il cui risultato e' di un tipo computazionale, p.e.
  $fix: \forall X.(M\ X \to M\ X) \to M\ X$

Essa e' particolamente utile per strutturare la semantica denotazionale dei linguaggi di programmazione. Infatti, M va' interpretato come il costruttore che fa' passare dal dominio/tipo dei valori τ al dominio/tipo $M\tau$ dei "programmi/computazioni che restituiscono valori in τ". In particolare, nel dare la traduzione $PL \longrightarrow ML_M(\Sigma)$ possiamo prescindere dalla esatta interpretazione di M, e quindi le scelte che nella semantica denotazione tradizionale (dove si deve definire una interpretazione di PL nella CCC $\mathcal{C}$) devono essere fatte in blocco, possono essere suddivise in due fasi (traduzione ed interpretazione) tra loro quasi del tutto indipendenti. Poiche' useremo frequentemente i tipi computazionali, introduciamo una notazione piu' conveniente di val_M e let_M (e spesso l'indice M sara' sottointeso):

- $[e]_M \stackrel{\Delta}{\equiv} val_M[\tau]\ e$, dove τ e' il tipo di e
  $$\text{val}\ \frac{\Gamma \vdash e : \tau}{\Gamma \vdash [e]_M : M\tau}$$
  Intuitivamente $[e]_M$ e' il programma che si limita a restituire il valore e
- $\mathsf{let}_M\, x \Leftarrow e_1 \,\mathsf{in}\, e_2 \stackrel{\Delta}{\equiv} let_M[\tau_1, \tau_2]\ (\lambda x : \tau_1.e_2)\ e_1$, dove $M\tau_1$ e' il tipo di e_1 e $M\tau_2$ e' il tipo di e_2.
  $$\text{let}\ \frac{\Gamma \vdash e_1 : M\tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : M\tau_2}{\Gamma \vdash \mathsf{let}_M\, x \Leftarrow e_1 \,\mathsf{in}\, e_2 : M\tau_2}$$
  Intuitivamente $\mathsf{let}_M\, x \Leftarrow e_1 \,\mathsf{in}\, e_2$ e' il programma che esegue e_1 per ottenere un valore di tipo τ_1 da legare a x, quindi esegue e_2 (che puo' fare riferimento ad x) per ottenere un lavore di tipi τ_2.
- $\mathsf{let}_M\, x_1 \Leftarrow e_1, \ldots, x_m \Leftarrow e_m \,\mathsf{in}\, e \stackrel{\Delta}{\equiv} \mathsf{let}_M\, x_1 \Leftarrow e_1 \,\mathsf{in}\, \ldots \mathsf{let}_M\, x_m \Leftarrow e_m \,\mathsf{in}\, e$

Diamo alcuni esempi di interpretazione di M nella categoria **Cpo** dei cpo e delle funzioni continue:

- $MX \stackrel{\Delta}{\equiv} X$, $[e]_M \stackrel{\Delta}{\equiv} e$ e $\mathsf{let}_M\, x \Leftarrow e_1 \,\mathsf{in}\, e_2 \stackrel{\Delta}{\equiv} (\lambda x : \tau_1.e_2)\ e_1$
- $MX \stackrel{\Delta}{\equiv} X + 1$, $[e]_M \stackrel{\Delta}{\equiv} in_0(e)$ e $\mathsf{let}_M\, x \Leftarrow e_1 \,\mathsf{in}\, e_2 \stackrel{\Delta}{\equiv} \mathsf{case}\ e_1\ \mathsf{of}\ [in_0(x) \Rightarrow e_2 \mid in_1(*) \Rightarrow in_1(*)]$
- $MX \stackrel{\Delta}{\equiv} X_\perp$, $[e]_M \stackrel{\Delta}{\equiv} \lfloor e \rfloor$ e $\mathsf{let}_M\, x \Leftarrow e_1 \,\mathsf{in}\, e_2 \stackrel{\Delta}{\equiv} \mathsf{let}\, x \Leftarrow e_1 \,\mathsf{in}\, e_2$ (per la notazione vedi Ch 8 in [Win93]).
  In questo caso si puo' interpretare fix, poiche' $X_\perp$ ha un elemente minimo.

Una interpretazione dei tipi computazionali deve soddisfare le seguenti equazioni tra termini ben tipati:

- $x : X,\ f : X \to MY \vdash \mathsf{let}\, x \Leftarrow [x] \,\mathsf{in}\, f\ x = f\ x : MY$
- $x' : MX \vdash \mathsf{let}\, x \Leftarrow x' \,\mathsf{in}\, [x] = x' : MX$
- $x' : MX,\ f : X \to MY,\ g : Y \to MZ \vdash \mathsf{let}\, y \Leftarrow (\mathsf{let}\, x \Leftarrow x' \,\mathsf{in}\, f\ x) \,\mathsf{in}\, g\ y = \mathsf{let}\, x \Leftarrow x',\ y \Leftarrow f\ x \,\mathsf{in}\, g\ y : MZ$

Diamo una formulazione equivalente delle equazioni, che prescindono dal tipaggio ed e' di piu' diretta applicazione

$$\mathsf{let}\, x \Leftarrow [e_1] \,\mathsf{in}\, e_2 = e_2[x := e_1] \tag{1}$$
$$\mathsf{let}\, x \Leftarrow e \,\mathsf{in}\, [x] = e \tag{2}$$
$$\mathsf{let}\, y \Leftarrow (\mathsf{let}\, x \Leftarrow e_1 \,\mathsf{in}\, e_2) \,\mathsf{in}\, e_3 = \mathsf{let}\, x \Leftarrow e_1,\ y \Leftarrow e_2 \,\mathsf{in}\, e_3 \qquad x \notin \mathrm{FV}(e_3) \tag{3}$$

Specifiche interpretazioni possono soddisfare altre equazioni (che nel caso di interpretazioni nella categoria **Cpo** dei cpo dovrebbero essere rimpiazzate da disequazioni $e_1 \leq e_2 : \tau$ tra termini ben-tipati).

HOFL	typed CBV translation $\_^e$	typed CBN translation $\_^l$
$\tau \in \mathsf{Type}$	$\tau^e \in \mathsf{Type}$	$\tau^l \in \mathsf{Type}$
int	N	N
$\tau_1 \times \tau_2$	$\tau_1^e \times \tau_2^e$	$M\tau_1^l \times M\tau_2^l$
$\tau_1 \to \tau_2$	$\tau_1^e \to M\tau_2^e$	$M\tau_1^l \to M\tau_2^l$
$t \in \mathsf{Exp}$	$t^e \in \mathsf{Exp}$	$t^l \in \mathsf{Exp}$
x	$[x]$	x
n	$[n]$	$[n]$
$t_1 + t_2$	$\mathsf{let}\, x_1 \Leftarrow t_1^e,\ x_2 \Leftarrow t_2^e \,\mathsf{in}\, [x_1 + x_2]$	$\mathsf{let}\, x_1 \Leftarrow t_1^l,\ x_2 \Leftarrow t_2^l \,\mathsf{in}\, [x_1 + x_2]$
(t_1, t_2)	$\mathsf{let}\, x_1 \Leftarrow t_1^e,\ x_2 \Leftarrow t_2^e \,\mathsf{in}\, [(x_1, x_2)]$	$[(t_1^l, t_2^l)]$
$\mathsf{fst}(t)$	$\mathsf{let}\, x \Leftarrow t^e \,\mathsf{in}\, [\pi_0(x)]$	$\mathsf{let}\, x \Leftarrow t^l \,\mathsf{in}\, \pi_0(x)$
$\lambda x{:}\tau.t$	$[\lambda x{:}\tau^e.t^e]$	$[\lambda x{:}M\tau^l.t^l]$
$t_1\ t_2$	$\mathsf{let}\, f \Leftarrow t_1^e,\ x \Leftarrow t_2^e \,\mathsf{in}\, f\ x$	$\mathsf{let}\, f \Leftarrow t_1^l \,\mathsf{in}\, f\ t_2^l$
$\mathsf{rec}\ f{:}\tau_1 \to \tau_2.\lambda x{:}\tau_1.t$	$fix(\lambda f'{:}M(\tau_1^e \to M\tau_2^e).[\lambda x{:}\tau_1^e.\mathsf{let}\, f \Leftarrow f' \,\mathsf{in}\, t^e])$	
$\mathsf{rec}\ x{:}\tau.t$		$fix(\lambda x{:}M\tau^l.t^l)$

Figure 1: Typed CBV/CBN Translations of HOFL

3.1 Traduzione typed CBV/CBN di HOFL

In questa sezione diamo due traduzione dei tipi e dei termini ben tipati del linguaggio HOFL (Ch 11 di [Win93]) in un metalinguaggio con tipi computazionali $ML_M(\Sigma)$, dove nella segnatura Σ si ha fix e

- una segnatura Σ_{int} per gli interi

  $\mathsf{N}{:}\,0$,
  $n{:}\,\mathsf{N}$ per ogni n intero,
  $op{:}\,\mathsf{N}^n \to \mathsf{N}$ per ogni op operazione aritmentica n-aria

Le traduzioni $\_^e$ per CBV e $\_^l$ per CBN sono definite per induzione sulla struttura dei tipi τ e dei termini t di HOFL

HOFL	$\_^e$	$\_^l$
$\tau \in \mathsf{Type}$	$\tau^e \in \mathsf{Type}$	$\tau^l \in \mathsf{Type}$
$t \in \mathsf{Exp}$	$t^e \in \mathsf{Exp}$	$t^l \in \mathsf{Exp}$
$\{x_i{:}\tau_i \mid i \in m\} \vdash t{:}\tau$	$\{x_i{:}\tau_i^e \mid i \in m\} \vdash t^e{:}M\tau^e$	$\{x_i{:}M\tau_i^l \mid i \in m\} \vdash t^l{:}M\tau^l$

La traduzione τ^* di un tipo τ di HOFL corrisponde al dominio dei valori di tipo τ. Inoltre la traduzione di $\Gamma \vdash t{:}\tau$ ci dice che in CBV le variabili variano sui valori, mentre in CBN variano sulle computazioni. A posteriori si dimostra

Proposition 3.1 *Se $\Gamma \vdash t{:}\tau$ e' derivabile, allora $(\Gamma \vdash t{:}\tau)^*$ e' derivabile (usando l'opportuna segnatura).*

Proof per induzione sulla derivazione di $\Gamma \vdash t{:}\tau$. ∎

Si osservi che nella traduzione CBV si impongono delle restrizioni sul costrutto $\mathsf{rec}\ x.t$, questo e' necessario perche' l'interpretazione CBV di $t \equiv 3$ e $\tau' \equiv (\lambda x.3)\ x$ sono uguali, mentre l'interpretazione di $\mathsf{rec}\ x.t$ e $\mathsf{rec}\ x.t'$ sono diverse: la prima e' $\lfloor 3 \rfloor$, la seconda e' $\perp$. Quindi non e' possibile dare una interpretazione *composizionale* di $\mathsf{rec}\ x.t$.

Una proprieta' fondamentale delle interpretazioni (viste a Logica) e' il lemma di sostituzione. Questo ha una controparte a livello di traduzioni.

Lemma 3.2 (Sostituzione per $\_^l$) $(t_2[x{:=}t_1])^l \equiv t_2^l[x{:=}t_1^l]$.

Proof per induzione sulla struttura di t_2. ∎

Nel caso della traduzione CBV il lemma di sostituzione vale solo quando si sostituisce ad una variabile un valore, inoltre per formularlo correttamente e' necessario introdurre una traduzione ausiliaria $\_^v$, che si applica ai soli valori per la semantica CBV (incluse le variabili)

$$x^v \stackrel{\Delta}{\equiv} x \qquad n^v \stackrel{\Delta}{\equiv} n \qquad (v_1, v_2)^v \stackrel{\Delta}{\equiv} (v_1^v, v_2^v) \qquad (\lambda x.t)^v \stackrel{\Delta}{\equiv} \lambda x{:}\tau^e.t^e$$

La traduzione ausiliaria soddisfa la seguente proprieta'

HOFL	untyped CBV translation $\_^e$	untyped CBN translation $\_^l$
$t \in \mathsf{Exp}$	$t^e \in \mathsf{Exp}$	$t^l \in \mathsf{Exp}$
x	$[x]$	x
n	$[intro\ in_0(n)]$	$[intro\ in_0(n)]$
$t_1 + t_2$	$\mathsf{let}\, intro\ in_0(x_1) \Leftarrow t_1^e,\ intro\ in_0(x_2) \Leftarrow t_2^e\, \mathsf{in}\, [intro\ in_0(x_1 + x_2)]$	*analogamente*
(t_1, t_2)	$\mathsf{let}\, x_1 \Leftarrow t_1^e,\ x_2 \Leftarrow t_2^e\, \mathsf{in}\, [intro\ in_1(x_1, x_2)]$	$[intro\ in_1(t_1^l, t_2^l)]$
$\mathsf{fst}(t)$	$\mathsf{let}\, intro\ in_1(x) \Leftarrow t^e\, \mathsf{in}\, [\pi_0(x)]$	$\mathsf{let}\, intro\ in_1(x) \Leftarrow t^l\, \mathsf{in}\, \pi_0(x)$
$\lambda x.t$	$[intro\ in_2(\lambda x{:}\, \mathsf{V}.t^e)]$	$[intro\ in_2(\lambda x{:}\, M\mathsf{V}.t^l)]$
$t_1\ t_2$	$\mathsf{let}\, intro\ in_2(f) \Leftarrow t_1^e,\ x \Leftarrow t_2^e\, \mathsf{in}\, f\ x$	$\mathsf{let}\, intro\ in_2(f) \Leftarrow t_1^l\, \mathsf{in}\, f\ t_2^l$
$\mathsf{rec}\ f.\lambda x.t$	$fix(\lambda f'{:}\, M\mathsf{V}.[intro\ in_2(\lambda x{:}\, \mathsf{V}.\mathsf{let}\, f \Leftarrow f'\, \mathsf{in}\, t^e)])$	
$\mathsf{rec}\ x.t$		$fix(\lambda x{:}\, M\mathsf{V}.t^l)$

Figure 2: Untyped CBV/CBN Translations of HOFL

Proposition 3.3 *Se $\Gamma \vdash v{:}\, \tau$ e' derivabile, allora $\Gamma^e \vdash v^v{:}\, \tau^e$ e' derivabile (usando l'opportuna segnatura).*

Proof per induzione sulla derivazione di $\Gamma \vdash v{:}\, \tau$. ∎

Inoltre il lemma di sostituzione per $\_^e$ vale solo a meno di uguaglianza indotta dagli assiomi per i tipi computazionali (ed altre proprieta' generali dell'uguaglianza che valgono in ogni CCC).

Lemma 3.4 (Sostituzione per $\_^e$) $v^e = [v^v]$ *e* $(t[x{:}= v])^e = t^e[x{:}= v^v]$.

Proof Il primo fatto e' per induzione su $v \in \mathsf{Val}$, mentre il secondo per induzione su $t \in \mathsf{Exp}$. ∎

3.2 Traduzione untyped CBV/CBN di HOFL

Se vogliamo dare la semantica denotazionale a qualunque termine t di HOFL, indipendententemente dal tipaggio, occorre definire una traduzione che mappa termini di HOFL in termini ben-tipati di $ML(\Sigma)$. Per far cio' e' necessario estendere la segnatura precedentemente considerata per definire la traduzione dei soli termini ben-tipati.

- dominio dei valori per interpretazione CBV non tipata $\mathsf{V} \cong \mathsf{N} + (\mathsf{V} \times \mathsf{V}) + (\mathsf{V} \to M\mathsf{V})$

  $\mathsf{V}{:}\, 0$, $intro{:}\, \mathsf{N} + (\mathsf{V} \times \mathsf{V}) + (\mathsf{V} \to M\mathsf{V}) \to \mathsf{V}$, $elim{:}\, \mathsf{V} \to \mathsf{N} + (\mathsf{V} \times \mathsf{V}) + (\mathsf{V} \to M\mathsf{V})$
- dominio dei valori per interpretazione CBN non tipata $\mathsf{V} \cong \mathsf{N} + (M\mathsf{V} \times M\mathsf{V}) + (M\mathsf{V} \to M\mathsf{V})$

  $\mathsf{V}{:}\, 0$, $intro{:}\, \mathsf{N} + (M\mathsf{V} \times M\mathsf{V}) + (M\mathsf{V} \to M\mathsf{V}) \to \mathsf{V}$, $elim{:}\, \mathsf{V} \to \mathsf{N} + (M\mathsf{V} \times M\mathsf{V}) + (M\mathsf{V} \to M\mathsf{V})$

Le equazioni di dominio per V non ammettono soluzione nella categoria **Set** degli insiemi (per ragioni di cardinalita'), tuttavia esse hanno soluzione nella categoria **Cpo** dei cpo, che fornisce un setting ideale per dare semantica alle definizioni ricorsive, sia morfismi, chedi oggetti (quest'ultimi a meno di isomorfismo). Per descrivere in modo piu' conciso la traduzione abbiamo introdotto la seguente notazione ausiliaria

- $\mathsf{let}\, intro\ in_i(x) \Leftarrow e_1\, \mathsf{in}\, e_2 \stackrel{\Delta}{\equiv} \mathsf{let}\, x' \Leftarrow e_1\, \mathsf{in}\, \mathsf{case}\ elim\ x'\ \mathsf{of}\ [in_i(x) \Rightarrow e_2 \mid \_ \Rightarrow \bot]$, dove $\bot = fix(\lambda x{:}\, MX.x){:}\, MX$

  $$\frac{\Gamma \vdash e_1{:}\, M\mathsf{V} \qquad \Gamma, x{:}\, \mathsf{V}_i \vdash e_2{:}\, M\tau_2}{\Gamma \vdash \mathsf{let}_M\, intro\ in_i(x) \Leftarrow e_1\, \mathsf{in}\, e_2{:}\, M\tau_2} \quad \text{dove } \mathsf{V} \cong \textstyle\coprod_i \mathsf{V}_i$$

A posteriori si dimostrano le seguenti proprieta' per le traduzioni non tipate:

Proposition 3.5 *Se $\mathrm{FV}(t) \subseteq \{x_i | i \in m\}$, allora $\{x_i{:}\, \mathsf{V} | i \in m\} \vdash t^e{:}\, M\mathsf{V}$ e $\{x_i{:}\, M\mathsf{V} | i \in m\} \vdash t^l{:}\, M\mathsf{V}$ sono derivabile (usando l'opportuna segnatura).*

Proof per induzione sulla struttura di t. ∎

Semantica denotazionale di HOFL con estensioni. Dare una semantica denotazione con errore di HOFL non tipato [si consiglia di usare $MX \stackrel{\Delta}{=} (X + 1)_\bot$]. Dare una semantica denotazione di HOFL tipato ed esteso con stato ed eccezioni, ma senza construtti new o newexc [si consiglia di usare $MX = S \to (X \times S)_\bot$ e $MX = (X + E)_\bot$].

HOFL con interactive input/output. Si consideri la seguente estensione di HOFL $t ::= \mathsf{get} \mid \mathsf{put}(e)$, dove get serve per leggere da input un intero, mentre $\mathsf{put}(e)$ scrive in output n e lo restituisce anche come valore, dove l'intero n e' il valore di e. Dare sia una semantica operazionale che denotazionale [DIFFICILE].

3.3 Teoria equazionale per il lambda-calcolo

Questa sezione descrive la teoria equazionale per il lambda-calcolo. Tale teoria e' sound e completa rispetto a qualunque interpretazione del lambda-calcolo in CCC, cioe' $\Gamma \vdash e_1 = e_2 : \tau$ e' vera in $\mathcal{C} \overset{\Delta}{\Longleftrightarrow} [\![\Gamma \vdash e_1 : \tau]\!] = [\![\Gamma \vdash e_2 : \tau]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$

- assiomi che valgono in qualunque CCC, oltre agli assiomi che asseriscono che l'uguaglianza e' una equivalenza ed una congruenza, si hanno gli assiomi

  $$\to.\beta\ \frac{\Gamma, x : \tau_1 \vdash e_2 : \tau_2 \qquad \Gamma \vdash e_1 : \tau_1}{\Gamma \vdash (\lambda x : \tau_1 . e_2)\ e_1 = e_2[x := e_1] : \tau_2} \qquad \to.\eta\ \frac{\Gamma \vdash e : \tau_1 \to \tau_2}{\Gamma \vdash e = (\lambda x : \tau_1 . e\ x) : \tau_1 \to \tau_2}\ x \notin \mathrm{FV}(e)$$

  $$\times.\beta\ \frac{\{\Gamma \vdash e_i : \tau_i | i \in n\}}{\Gamma \vdash \pi_i(e_i | i \in n) = e_i : \tau_i}\ i \in n \qquad \times.\eta\ \frac{\Gamma \vdash e : (\tau_i | i \in n)}{\Gamma \vdash e = (\pi_i(e) | i \in n) : (\tau_i | i \in n)}$$

  nelle regole di congruenza per binder, come la λ-astrazione, si ha una modifica del contesto, p.e.

  $$\to.\beta\ \frac{\Gamma, x : \tau_1 \vdash e_1 = e_2 : \tau_2}{\Gamma \vdash (\lambda x : \tau_1 . e_1) = (\lambda x : \tau_1 . e_2) : \tau_1 \to \tau_2}$$

- assiomi che valgono in qualunque CCC con somme

  $$+.\beta\ \frac{\Gamma \vdash e : \tau_i \qquad \{\Gamma, x_i : \tau_i \vdash e_i : \tau | i \in n\}}{\Gamma \vdash (\mathsf{case}\ in_i(e)\ \mathsf{of}\ [in_i(x_i) \Rightarrow e_i | i \in n]) = e_i[x_i := e] : \tau}\ i \in n$$

  $$+.\eta\ \frac{\Gamma \vdash e : [\tau_i | i \in n] \qquad \Gamma, x : [\tau_i | i \in n] \vdash e' : \tau}{\Gamma \vdash e'[x := e] = \mathsf{case}\ e\ \mathsf{of}\ [in_i(x_i) \Rightarrow e'[x := in_i(x_i)] | i \in n] : \tau}$$

- assiomi per i tipi computazionali

  $$M.\beta\ \frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : M\tau_2}{\Gamma \vdash (\mathsf{let}_M\ x \Leftarrow [e_1]_M\ \mathsf{in}\ e_2) = e_2[x := e_1] : M\tau_2} \qquad M.\eta\ \frac{\Gamma \vdash e : M\tau}{\Gamma \vdash e = (\mathsf{let}_M\ x \Leftarrow e\ \mathsf{in}\ [x]_M) : M\tau}$$

  $$M.\mathrm{ass}\ \frac{\Gamma \vdash e_1 : M\tau_1 \qquad \Gamma, x_1 : \tau_1 \vdash e_2 : M\tau_2 \qquad \Gamma, x_2 : \tau_2 \vdash e_3 : M\tau_3}{\Gamma \vdash (\mathsf{let}_M\ x_2 \Leftarrow (\mathsf{let}_M\ x_1 \Leftarrow e_1\ \mathsf{in}\ e_2)\ \mathsf{in}\ e_3) = (\mathsf{let}\ x_1 \Leftarrow e_1,\ x_2 \Leftarrow e_2\ \mathsf{in}\ e_3) : M\tau_3}\ x_1 \notin \mathrm{FV}(e_3)$$

- assiomi per operatori di punto fisso

  $$\mathrm{fix}\ \frac{\Gamma \vdash e : M\tau \to M\tau}{\Gamma \vdash fix\ e = e\ (fix\ e) : M\tau}$$

E' possibile dare una teoria con disequazioni che e' piu' accurata (soprattutto in relazione all'operatore di punto fisso) per l'interpretazione del lambda-calcolo nella categoria **Cpo** dei cpo. Una disequazione $\Gamma \vdash e_1 \leq e_2 : \tau$ e' vera in **Cpo** $\overset{\Delta}{\Longleftrightarrow} [\![\Gamma \vdash e_1 : \tau]\!] \leq [\![\Gamma \vdash e_2 : \tau]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$, dove l'ordine sui morfismi e' definito in modo puntuale.

- assiomi per operatori di minimo punto fisso

  $$\mathrm{fix.1}\ \frac{\Gamma \vdash e : M\tau \to M\tau}{\Gamma \vdash e\ (fix\ e) \leq fix\ e : M\tau} \qquad \mathrm{fix.2}\ \frac{\Gamma \vdash e\ e' \leq e' : M\tau}{\Gamma \vdash fix\ e \leq e' : M\tau}$$