Appunti del corso di

Tipi e Linguaggi di Programmazione

del Prof. Simone Martini

(ultima revisione: 8 settembre 2010)

Andrea Simonetto
Università di Bologna
simonett [at] cs [dot] unibo [dot] it
http://simonett.web.cs.unibo.it/

Introduzione

Questi sono gli appunti del corso di *tipi e linguaggi di programmazione* tenuto dal Professor Simone Martini durante l'anno accademico 2008/09. Non hanno la pretesa di essere esaustivi e contengono certamente qualche errore, per cui sono benvenuti eventuali commenti, critiche e correzioni. Non sono da intendere come riferimento conclusivo o sostitutivo al testo seguito dal professore, né tantomeno alla frequentazione delle lezioni, bensì come un supporto allo studio; mi auguro che vi sia utile.

Il corso, tratto prevalentemente da Pierce [2002], consiste in una trattazione matematico–formale dei fondamenti di alcuni sistemi di tipo utilizzati nei linguaggi di programmazione e delle proprietà di cui godono.

Le motivazioni che animano questa disciplina sono molteplici; tra le principali osserviamo che i tipi nei linguaggi di programmazione:

- agevolano la *progettazione* di sistemi software complessi;
- garantiscono *proprietà di correttezza* dei programmi;
- permettono *implementazioni efficienti* di interpreti e compilatori;

L'aspetto di progettazione non sarà per noi oggetto di studio, ma è bene ricordare che i sistemi di tipo forniscono un potente strumento di modellazione per descrivere i concetti del dominio d'interesse (pensiamo ad esempio ai linguaggi orientati agli oggetti). Inoltre i tipi permettono di controllare alcune relazioni espresse a livello concettuale (come ad esempio la relazione di ereditarietà). Da qui si evince immediatamente l'importanza di dare solide basi formali a questo strumento così irrinunciabile nella pratica informatica odierna.

Le proprietà di correttezza sono invece il tema centrale del corso: cosa si intende per correttezza rispetto al sistema di tipo? Come inscrivere le nostre intuizioni in un contesto formale? Cosa è possibile garantire e quali

sono invece le limitazioni? Queste sono le domande a cui tenteremo di dare risposta.

Una nozione trasversale a tutto lo svolgimento del corso sarà la proprietà di *type safety* (o sicurezza rispetto ai tipi), di cui godono alcuni linguaggi di programmazione tipizzati, e che possiamo informalmente riassumere dicendo che:

> un linguaggio è *type safe* se data *un'espressione ben tipizzata*, la sua *valutazione* termina correttamente su un *valore* del tipo opportuno oppure determina un *errore* che viene "intrappolato" dal *sistema di tipo*.

Osserviamo che in questa definizione alcuni concetti sono lasciati all'intuzione: in seguito ci occuperemo di formalizzare e dettagliare opportunamente tali nozioni. Proseguendo con l'analisi della *type safety*, essa si può considerare scomposta in due proprietà indipendenti come segue:

$$\text{Type safety} = \text{Progresso} + \text{Preservazione}$$

La *preservazione* garantisce che il tipo di ogni espressione verrà preservato durante tutta l'esecuzione, mentre il *progresso* asserisce che ogni espressione ben tipizzata o è un valore o può essere ulteriormente ridotta, cioè la valutazione di termini ben tipizzati non è mai "bloccata".

Il linguaggio di riferimento sarà il λ-calcolo, in quanto è lo strumento più adatto per lo studio dei fondamenti formali dei linguaggi di programmazione, a cui accosteremo sistemi di tipo via via più sofisticati al fine di ottenere maggiore espressività e proprietà di crescente interesse, esaminando nell'ordine: un linguaggio giocattolo per definire quello che sarà il nostro *modus operandi* nel primo capitolo; i *tipi semplici* e le relative *estensioni* nel secondo e nel terzo capitolo; la *relazione di sottotipo* e i *tipi ricorsivi* nel quarto ed il *polimorfismo* nel quinto ed ultimo capitolo.

Indice

1 Espressioni aritmetiche e booleane **1**
- 1.1 Normalizzazione . . . 2
- 1.2 Valori . . . 4
- 1.3 Tipi . . . 4
- 1.4 Type safety . . . 5

2 Lambda calcolo tipizzato semplice **9**
- 2.1 Esempi . . . 10
- 2.2 La corrispondenza di Curry-Howard . . . 12
- 2.3 Safety . . . 13
- 2.4 Cancellazione del tipo . . . 17
- 2.5 Normalizzazione . . . 18
  - 2.5.1 Normalizzazione debole . . . 19
  - 2.5.2 Normalizzazione forte . . . 21
- 2.6 Espressività del calcolo . . . 23
- 2.7 Ricostruzione del tipo . . . 25

3 Estensioni del calcolo **33**
- 3.1 Ambienti locali . . . 33
- 3.2 Il tipo singoletto . . . 34
- 3.3 Operatore di sequenza . . . 35
- 3.4 Annotazioni esplicite di tipo . . . 36
- 3.5 Prodotti e somme . . . 36
- 3.6 Record . . . 38
- 3.7 Varianti . . . 39
- 3.8 Liste . . . 40
- 3.9 Ricorsione generale . . . 42
- 3.10 Locazioni . . . 43

3.11 Errori ed eccezioni 46

4 Sottotipi **49**
4.1 Proprietà 52
4.1.1 Un sistema ristretto 57
4.2 Tipi ricorsivi 61

5 Polimorfismo parametrico esplicito **69**
5.1 Proprietà 71
5.2 Quantificatore esistenziale 73
5.3 Ricostruzione del tipo 73

A Correttezza e completezza dell'algoritmo di subtyping per tipi ricorsivi (M. Patrignani) **75**

Bibliografia **81**

Capitolo 1

Espressioni aritmetiche e booleane

Come primo caso di studio, definiamo un semplice linguaggio per il calcolo di espressioni aritmetiche e booleane, che servirà ad introdurre le tecniche di base che useremo nei capitoli successivi. Useremo i caratteri *corsivi* per denotare i simboli del metalinguaggio di definizione. La grammatica BNF del nostro semplice linguaggio è la seguente:

$$t ::= \texttt{tt} \mid \texttt{ff} \mid \texttt{if } t \texttt{ then } t \texttt{ else } t \mid \texttt{0} \mid \texttt{succ } t \mid \texttt{pred } t \mid \texttt{iszero } t$$

Ci concentriamo inizialmente sul sottolinguaggio delle espressioni booleane (`tt`, `ff`, `if` t `then` t `else` t). La semantica è data dall'operazione di *riduzione ad un passo* applicata ad un termine. L'operazione di *riduzione* o *riscrittura* mappa termini in termini, ed è definita dalle seguenti regole operazionali:

$$\texttt{if tt then } t_2 \texttt{ else } t_3 \to t_2 \qquad (\textsc{If}^{\textsc{tt}})$$

$$\texttt{if ff then } t_2 \texttt{ else } t_3 \to t_3 \qquad (\textsc{If}^{\textsc{ff}})$$

$$\frac{t_1 \to t_1'}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \to \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3} \qquad (\textsc{If})$$

Inoltre $\to^*$ è la *chiusura riflessiva e transitiva* di $\to$. Si osserva che la regola (If) è semplicemente la *chiusura per contesti* delle regole ($\textsc{If}^{\textsc{tt}}$) e ($\textsc{If}^{\textsc{ff}}$). È possibile dimostrare che, per questo semplice calcolo, la riduzione ad un passo è deterministica. Ci restringiamo nuovamente al sottolinguaggio delle espressioni booleane.

Teorema 1.1 ($\to$ è deterministica)**.** *Se $t \to t'$ e $t \to t''$ allora $t' \equiv t''$ (dove $\equiv$ è la relazione di* equivalenza sintattica *fra termini).*

Dimostrazione. Per induzione sulla derivazione di $t \to t'$.

- Casi base.

  (i) $\texttt{if tt then } t_2 \texttt{ else } t_3 \to t_2$

  (ii) $\texttt{if ff then } t_2 \texttt{ else } t_3 \to t_3$

  In questi due casi non esistono altre regole per ridurre t, perciò l'asserto è vero.

- Caso induttivo: $t \equiv \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3$. Produciamo due diverse derivazioni per $t \to t'$ come segue:

  a)
  $$\frac{\begin{array}{c}\vdots \\ t_1 \to t_1'\end{array}}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \to \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3}$$

  b)
  $$\frac{\begin{array}{c}\vdots \\ t_1 \to t_1''\end{array}}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \to \texttt{if } t_1'' \texttt{ then } t_2 \texttt{ else } t_3}$$

  Per ipotesi induttiva, sappiamo che se $t_1 \to t_1'$ e $t_1 \to t_1''$ allora $t_1' \equiv t_1''$. Perciò le due derivazioni sopra sono in realtà la stessa derivazione, da cui si conclude che l'asserto vale anche nel caso induttivo.

□

Per poter parlare di computazione, oltre alla regola di riduzione, abbiamo bisogno del concetto di *forma normale*, che raccolga l'insieme di quei termini su cui nessuna ulteriore riduzione è possibile.

Definizione 1.2. *Un termine t è in* forma normale *(o t è una f.n.) sse non esiste alcun termine t' tale che $t \to t'$.*

1.1 Normalizzazione

Si osserva che le questo semplice calcolo è sempre terminante: questo fatto è catturato dal *teorema di normalizzazione.*

Teorema 1.3 (normalizzazione)**.** *Per ogni termine t, esiste un termine t' f.n. tale che $t \to^* t'$.*

Dimostrazione. Se t è già in forma normale il teorema vale banalmente, in quanto $\to^*$ è riflessiva; altrimenti la dimostrazione procede assegnando un peso (in forma di numero naturale) ai termini e dimostrando che in una

catena di riduzioni il peso decresce strettamente. In particolare, sia w la funzione di peso, definita come segue:

$$\begin{aligned} w(\texttt{tt}) &\triangleq 1 \\ w(\texttt{ff}) &\triangleq 1 \\ w(\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3) &\triangleq 1 + w(t_1) + w(t_2) + w(t_3) \end{aligned}$$

Si vuole dimostrare che:

$$\text{se } t \to t' \text{ allora } w(t) > w(t')$$

Per induzione sulla derivazione di $t \to t'$:

- Casi base.

  (i) $t \equiv \texttt{if tt then } t_2 \texttt{ else } t_3 \quad \to \quad t' \equiv t_2$.

  $$\begin{aligned} w(t) > w(t') \quad &\text{sse} \quad 1 + w(\texttt{tt}) + w(t_2) + w(t_3) > w(t_2) \\ &\text{sse} \quad 2 + w(t_3) > 0 \end{aligned}$$

  che è sempre vero, dato che w ha per codominio i numeri naturali.

  (ii) $t \equiv \texttt{if ff then } t_2 \texttt{ else } t_3 \quad \to \quad t' \equiv t_3$. Analogo al caso precedente.

- Caso induttivo.

  $$\frac{\begin{array}{c}\vdots \\ t_1 \to t_1'\end{array}}{\texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 \to \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3}$$

  $$\begin{aligned} w(t) > w(t') \quad &\text{sse} \quad 1 + w(t_1) + w(t_2) + w(t_3) > 1 + w(t_1') + w(t_2) + w(t_3) \\ &\text{sse} \quad w(t_1) > w(t_1') \end{aligned}$$

  che è vero per ipotesi induttiva.

È immediato estendere questo risultato ad ogni passo di una catena di riduzioni non vuota:

$$t \equiv t_0 \to t_1 \to t_2 \to \ldots \to t_i \to \ldots \qquad (\text{con } i > 0)$$

in cui ad ogni riduzione la lunghezza del termine descresce e che pertanto deve terminare dopo un certo numero n di passi su un termine $t_n \equiv t'$ su cui nessuna ulteriore riduzione è applicabile (cioè t' forma normale). □

1.2 Valori

Allarghiamo ora il nostro punto di vista, considerando il calcolo nella sua interezza: notiamo subito che ci sono forme normali del tipo:

$$\texttt{pred succ tt}$$

a cui non siamo interessati, che anzi, costituiscono un effetto indesiderato della computazione. Quello che vorremmo è definire una categoria di termini "legali", in qualche modo "sensati", quelli che siamo interessati ad osservare, i cosiddetti *valori*. Nel nostro calcolo, i *valori* sono:

$$\begin{array}{rcl} vb & ::= & \texttt{tt} \mid \texttt{ff} \\ vn & ::= & \texttt{0} \mid \texttt{succ}\ vn \end{array}$$

dove vb sta per "valori booleani", mentre vn per "valori naturali". La differenza tra valori e forme normali è che i valori costituiscono una sorta di "categoria sintattica", che non si basa sulla regola di riduzione ma solo sulla struttura dei termini. Intuitivamente, i valori sono "gli osservabili" del nostro linguaggio, cioè quei termini che, da un punto di vista di semantica intesa, sono risultati di computazioni che terminano senza errori. Per contro, un termine che è in forma normale ma non è un valore si dice *bloccato*.

Per concludere, estendiamo la semantica del calcolo con le regole strutturali per le espressioni aritmetiche:

$$\frac{t \to t'}{\texttt{succ}\ t \to \texttt{succ}\ t'}\ (\textsc{Esucc})$$

$$\begin{array}{rll} \texttt{pred}\ 0 \to 0 & & (\textsc{Epred}^0) \\ \texttt{pred succ}\ vn \to vn & & (\textsc{Epred}^{>0}) \\ \texttt{iszero}\ 0 \to\ \texttt{tt} & & (\textsc{Eiszero}^0) \\ \texttt{iszero succ}\ vn \to\ \texttt{ff} & & (\textsc{Eiszero}^{>0}) \end{array}$$

$$\frac{t \to t'}{\texttt{pred}\ t \to \texttt{pred}\ t'}\ (\textsc{Epred}) \qquad \frac{t \to t'}{\texttt{iszero}\ t \to \texttt{iszero}\ t'}\ (\textsc{Eiszero})$$

Nuovamente (Epred) e (Eiszero) sono rispettivamente le chiusure per contesti delle regole per `pred` e `iszero`.

1.3 Tipi

I termini bloccati corrispondono a quelli senza significato o errati, come ad esempio `pred ff`. Vorremmo restringere il linguaggio per escludere ogni

termine la cui valutazione si fermerà su un termine bloccato. Per fare ciò, ci serve la capacità di distinguere tra termini che ridurranno ad un valore numerico (siccome saranno gli unici che potranno comparire come argomenti di `pred`, `succ` e `iszero`) ed i termini che ridurranno a booleani. Introduciamo quindi due tipi, `Nat` e `Bool`, per effettuare questa classificazione. Le metavariabili $S, T, U, \ldots$ saranno usate in seguito per denotare i tipi.

L'obiettivo è arrivare ad affermare che la frase "un termine t ha tipo T" (o "t appartiene a T", o anche "t è un elemento di T") significa che t valuterà sempre ad un valore di tipo T. Questo giudizio può essere dato "staticamente", cioè senza alcuna valutazione di t. Per contro, un termine la cui valutazione risulta bloccata non potrà essere tipizzato.

Il *sistema di tipi* di un linguaggio è dato dalla sintassi dei tipi, nel nostro caso:

$$T ::= \texttt{Bool} \mid \texttt{Nat}$$

e dalle regole di tipizzazione o *giudizi di tipo*:

$$\vdash \texttt{tt} : \texttt{Bool} \quad (\textsc{Ttt})$$
$$\vdash \texttt{ff} : \texttt{Bool} \quad (\textsc{Tff})$$
$$\vdash \texttt{0} : \texttt{Nat} \quad (\textsc{Tzero})$$

$$\frac{\vdash t_1 : \texttt{Bool} \qquad \vdash t_2 : T \qquad \vdash t_3 : T}{\vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : T} \ (\textsc{Tif}) \qquad \frac{\vdash t : \texttt{Nat}}{\vdash \texttt{succ } t : \texttt{Nat}} \ (\textsc{Tsucc})$$

$$\frac{\vdash t : \texttt{Nat}}{\vdash \texttt{pred } t : \texttt{Nat}} \ (\textsc{Tpred}) \qquad \frac{\vdash t : \texttt{Nat}}{\vdash \texttt{iszero } t : \texttt{Bool}} \ (\textsc{Tiszero})$$

1.4 Type safety

La *safety* (a volte chiamata anche *type soundness*) è la proprietà più importante dei sistemi di tipo: i termini ben tipizzati non "finiscono male". Come abbiamo già sottolineato, un termine "finisce male" nella computazione quando riduce ad un termine bloccato, cioè un termine che non è un valore ma su cui non si possono applicare le regole di riduzione. La *safety* ci dice che se un termine è ben tipizzato, allora non si blocca. La *safety* si può esprimere come coppia di proprietà:

- *Progresso*: un termine ben tipizzato non è bloccato (cioè è un valore o può effettuare un passo di riduzione). Formalmente:

$$\text{se } \vdash t : T \text{ allora } t \text{ è un valore, oppure } t \to t'$$

- *Preservazione* (o *subject reduction*): se un termine ben tipizzato può effettuare un passo di riduzione, allora il termine risultante dalla riduzione è ben tipizzato. Formalmente:

$$\text{se } \vdash t : T \text{ e } t \to t' \text{ allora } \vdash t' : T$$

La *safety* è un ovvio corollario delle due proprietà sopra: "ogni termine tipizzabile riduce ad un valore". Riassumento:

$$\text{Type safety} = \text{Progresso} + \text{Preservazione}$$

Per provare *progresso* e *preservazione* bisogna introdurre alcuni lemmi e concetti.

Lemma 1.4 (Inversione)**.**

se $\vdash \texttt{tt} : T$ *allora* $T \equiv \texttt{Bool}$
se $\vdash \texttt{ff} : T$ *allora* $T \equiv \texttt{Bool}$
se $\vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : T$ *allora* $\vdash t_1 : \texttt{Bool}$, $\vdash t_2 : T$, $\vdash t_3 : T$
se $\vdash \texttt{0} : T$ *allora* $T \equiv \texttt{Nat}$
se $\vdash \texttt{succ } t : T$ *allora* $T \equiv \texttt{Nat}$, $\vdash t : \texttt{Nat}$
se $\vdash \texttt{pred } t : T$ *allora* $T \equiv \texttt{Nat}$, $\vdash t : \texttt{Nat}$
se $\vdash \texttt{iszero } t : T$ *allora* $T \equiv \texttt{Bool}$, $\vdash t : \texttt{Nat}$

Dimostrazione. Immediata, per ispezione delle regole. □

Teorema 1.5 (Unicità del tipo)**.**

se $\vdash t : T$ *e* $\vdash t : R$ *allora* $T \equiv R$

Dimostrazione. Per induzione sulla derivazione di $\vdash t : T$ usando il lemma d'inversione. □

Lemma 1.6 (Forme canoniche)**.**

se $\vdash v : \texttt{Bool}$ *e* v *è un valore* *allora* $v \equiv \texttt{tt}$ *oppure* $v \equiv \texttt{ff}$
se $\vdash v : \texttt{Nat}$ *e* v *è un valore* *allora* $v \equiv \texttt{0}$ *oppure* $v \equiv \texttt{succ } vn$

per qualche vn *valore naturale.*

Dimostrazione. Immediata, via il lemma d'inversione. □

Teorema 1.7 (Progresso)**.**

$$\textit{se} \quad \vdash t : T \quad \textit{allora} \quad t \textit{ è un valore oppure } t \to t' \textit{ per qualche } t'$$

Dimostrazione. Per induzione sulla derivazione di $\vdash t : T$.

- Casi base.
  - (i) se $\vdash \texttt{tt} : \texttt{Bool}$ allora $\texttt{tt}$ è un valore;
  - (ii) se $\vdash \texttt{ff} : \texttt{Bool}$ allora $\texttt{ff}$ è un valore;
  - (iii) ...
- Casi induttivi.
  - (i) $t \equiv \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3$

$$\frac{\begin{array}{ccc}\vdots & \vdots & \vdots \\ \vdash t_1 : \texttt{Bool} & \vdash t_2 : T & \vdash t_3 : T\end{array}}{\vdash \texttt{if } t_1 \texttt{ then } t_2 \texttt{ else } t_3 : T}$$

    Per ipotesi induttiva, t_1 può:
    - essere un valore, nel qual caso, per il lemma 1.6 (*forme canoniche*), o è $t_1 \equiv \texttt{tt}$ o $t_1 \equiv \texttt{ff}$. In ambedue i casi, un passo di riduzione per t è possibile (se $t_1 \equiv \texttt{tt}$ allora $t \to t_2$, altrimenti, se $t_1 \equiv \texttt{ff}$ allora $t \to t_3$);
    - ridurre a qualche t_1', cioè $t_1 \to t_1'$, per cui si ha che:

$$t \to \texttt{if } t_1' \texttt{ then } t_2 \texttt{ else } t_3$$

  - (ii) Altri casi analoghi.

□

Teorema 1.8 (Preservazione o Subject reduction)**.**

$$\textit{se} \quad \vdash t : T \quad e \quad t \to t' \quad \textit{allora} \quad \vdash t' : T$$

Dimostrazione. Per induzione sulla derivazione di $\vdash t : T$.

- Casi base.
  - (i) $\vdash \texttt{0} : \texttt{Nat}$, ma non esiste alcun t' tale che $\texttt{0} \to t'$, per cui l'asserto è vacuamente vero;
  - (ii) gli altri casi base procedono con la stessa argomentazione.

- Casi induttivi.

  (i) $t \equiv \texttt{succ}\ t_1$.

  $$\frac{\vdash t_1 : \texttt{Nat}}{\vdash \texttt{succ}\ t_1 : \texttt{Nat}} \qquad \frac{t_1 \to t_1'}{\texttt{succ}\ t_1 \to \texttt{succ}\ t_1'}$$

  Per ipotesi induttiva:

  $$\text{se} \quad \vdash t_1 : \texttt{Nat} \quad \text{e} \quad t_1 \to t_1' \quad \text{allora} \quad \vdash t_1' : \texttt{Nat}$$

  e quindi:

  $$\frac{\vdash t_1' : \ \texttt{Nat}}{\vdash \texttt{succ}\ t_1' : \texttt{Nat}}$$

  (ii) Altri casi per induzione diretta.

□

Capitolo 2

Lambda calcolo tipizzato semplice

Introduciamo il formalismo di base della famiglia dei linguaggi con tipi, noto come *lambda calcolo tipizzato semplice* (o *simply typed lambda calculus*) di Church [1940] e Curry [1958]. Non ci addentreremo nello studio del lambda calcolo puro (per un testo introduttivo al lambda calcolo, vedi Hindley and Seldin [1986], per un manuale completo, vedi Barendregt [1981]).

Quello che vogliamo ottenere è un linguaggio tipizzato per combinare i booleani (e potenzialmente altri tipi di base) e le primitive del lambda calcolo puro. In pratica si tratta di elaborare un sistema di tipi per le variabili, le applicazioni e le astrazioni che mantenga la proprietà di safety, pur non restringendo eccessivamente il linguaggio (cioè tale da permetterci di tipizzare la maggior parte di programmi a cui siamo interessati).

La sintassi del calcolo è quella usuale, con l'aggiunta di un'informazione inerente al *tipo dell'argomento* delle astrazioni (*annotazione di tipo*).

Definizione 2.1 (Sintassi)**.** *Sia x una metavariabile appartenente ad un insieme enumerabile di nomi:*

$$t ::= x \mid t\ t \mid \lambda x : T.t$$

dove T è il tipo dell'argomento del λ, formalmente:

$$T ::= o \mid T \to T \qquad con \qquad o \in \{\texttt{Bool}, \ldots\}$$

cioè, l'argomento di una λ-astrazione può essere un oggetto appartenente ad un tipo base oppure una funzione ($T \to T$). Il costruttore di tipo $\to$ associa a destra – cioè, l'espressione $T_1 \to T_2 \to T_3$ sta per $T_1 \to (T_2 \to T_3)$.

La semantica del calcolo è fornita dalle regole di riduzione ($\rightarrow$). Il concetto di forma normale è del tutto analogo a quello visto in precedenza: un termine t è una *forma normale* sse non esiste alcun termine t' tale che $t \rightarrow t'$. I *valori*, cioè le forme normali "sensate", quelle che siamo interessati ad osservare, sono le λ-astrazioni:

$$v ::= \lambda x : T.t$$

Definizione 2.2 (Weak call-by-value β-reduction)**.** *Siano v_1, v_2 valori:*

$$(\lambda x : T.t_1)v_2 \rightarrow t_1[v_2/x] \quad (\mathrm{E}\beta)$$

$$\frac{t_1 \rightarrow t_1'}{t_1 t_2 \rightarrow t_1' t_2}\,(\mathrm{E}\beta^{LX}) \qquad \frac{t_2 \rightarrow t_2'}{v_1 t_2 \rightarrow v_1 t_2'}\,(\mathrm{E}\beta^{RX})$$

Si osserva che manca la regola di chiusura per le λ-astrazioni: infatti, in questo calcolo, non si riduce mai sotto un λ.

Definizione 2.3 (Contesti o basi)**.** *Un contesto serve ad associare alle variabili libere di un'espressione il tipo corrispondente. Formalmente, i contesti sono nella forma:*

$$\Gamma ::= \epsilon \mid \Gamma,\ x : T$$

dove ϵ è il contesto vuoto. In pratica, sono liste di coppie (Variabile, Tipo).

A questo punto abbiamo tutti gli ingredienti per definire le regole di tipizzazione dei λ-termini:

Definizione 2.4.

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T}\,(\mathrm{T}_{\mathrm{VAR}}) \qquad \frac{\Gamma, x : T \vdash t : S}{\Gamma \vdash \lambda x : T.t : T \rightarrow S}\,(\mathrm{T}_{\mathrm{ABS}})$$

$$\frac{\Gamma \vdash t_1 : T \rightarrow S \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1 t_2 : S}\,(\mathrm{T}_{\mathrm{APP}})$$

2.1 Esempi

Seguono alcuni esempi e considerazioni sui termini del lambda calcolo tipizzato semplice ed alcune dimostrazioni di corretta tipizzazione degli stessi:

1. Identità:

$$\vdash \lambda x : T.x \ : T \rightarrow T \qquad \frac{x : T \vdash x : T}{\vdash \lambda x : T.x : T \rightarrow T}$$

2. Versioni tipizzate dei *numerali di Church* (n^T):

$$\begin{array}{lcll} 0^T & \triangleq & \lambda f : T.\lambda x : S.x & : T\to S\to S \\ 1^T & \triangleq & \lambda f : T\to S.\lambda x : T.fx & : (T\to S)\to T\to S \\ 2^T & \triangleq & \lambda f : T\to T.\lambda x : T.f(fx) & : (T\to T)\to T\to T \end{array}$$

Si osserva che queste tre varianti dei numerali di Church hanno tipi diversi. Tuttavia è possibile riscrivere 0^T e 1^T in modo che abbiano tutti e tre lo stesso tipo:

$$\begin{array}{lcll} 0^T & \triangleq & \lambda f : T\to T.\lambda x : T.x & : (T\to T)\to T\to T \\ 1^T & \triangleq & \lambda f : T\to T.\lambda x : T.fx & : (T\to T)\to T\to T \end{array}$$

In generale:

$$\begin{array}{lcl} n^T & : & (T\to T)\to T\to T \\ n^T & \triangleq & \lambda f : T\to T.\lambda x : T.f^{\underline{n}}x \end{array}$$

3. Consideriamo il seguente esempio:

$$\begin{array}{rcl} x : o,\ y : o & \nvdash & xy \\ x : o\to o,\ y : o & \vdash & xy \end{array}$$

Si osserva che lo stesso λ-termine può essere correttamente o scorrettamente tipizzato *in funzione del contesto*.
Problema: esistono Γ e T tali che:

$$\Gamma \vdash xx : T \quad ?$$

No. Infatti:

$$\dfrac{\dfrac{x : T\to T \in \Gamma}{\Gamma \vdash x : T\to T} \quad \dfrac{x : T \in \Gamma}{\Gamma \vdash x : T}}{\Gamma \vdash xx : T}$$

ma in Γ, la variabile libera x non può contemporaneamente appartenere a due tipi diversi!

4. Termine **S** (o *combinatore* **S**):

$$\vdash \lambda x : T\to S\to R.\lambda y : T\to S.\lambda z : T.xz(yz) \ : (T\to S\to R)\to(T\to S)\to T\to R$$

2.2 La corrispondenza di Curry-Howard

Si osserva che, se si interpretano i tipi come proposizioni logiche (dove $\to$ è l'implicazione logica), tutti i tipi visti fin qui sono tautologie. Questo ci suggerisce che deve esistere una qualche correlazione tra le regole di tipizzazione viste finora e quelle della logica proposizionale relative all'implicazione.

Osserviamo il costruttore di tipo $\to$: questo è corredato di due regole di tipizzazione:

1. una regola di *introduzione* (TABS) che descrive come gli elementi di tipo "freccia" devono essere creati;

2. una regola di *eliminazione* (TAPP) che descrive come quegli elementi devono essere usati.

Quando una forma d'introduzione (λ-astrazione) è un sottotermine immediato di una forma d'eliminazione (applicazione), il risultato è un redex. Le regole di tipo per il lambda calcolo tipizzato semplice "diventano" le regole della deduzione naturale per l'operatore $\supset$ (l'implicazione) non appena si "scordino" i termini.

La terminologia di "introduzione/eliminazione" nasce da una connessione tra la teoria dei tipi e la logica, nota come *corrispondenza o isomorfismo di Curry–Howard* (Curry [1958]; Howard [1980]). L'idea è che, nelle logiche costruttive, una dimostrazione di una proposizione P consiste in *una concreta evidenza di* P. Quello che Curry e Howard notarono è che questa evidenza ha una forte controparte computazionale. Ad esempio, una dimostrazione di una proposizione $P \supset Q$ (P implica Q) può essere vista come una procedura meccanica che, data una dimostrazione per P, costruisce una dimostrazione per Q. Analogamente, una dimostrazione per $P \wedge Q$ consiste in una dimostrazione di P insieme ad una per Q. Questa osservazione dà luogo alla corrispondenza tra logica e linguaggi di programmazione riassunta nella seguente tabella:

Logica	**Linguaggi di programmazione**
Proposizioni	Tipi
Proposizione $P \supset Q$	Tipo $P \to Q$
Dimostrazione della proposizione P	Termine t di tipo P
La proposizione P è dimostrabile	Il tipo P è abitato (da qualche termine)

È possibile dimostrare che un termine del lambda calcolo tipizzato semplice è una dimostrazione della proposizione logica corrispondente al suo tipo.

Consideriamo la principale regola di computazione:

$$(\lambda x : T.t_1)t_2 \to t_1[t_2/x]$$

Il redex a sinstra della freccia viene tipizzato come segue:

$$\dfrac{\dfrac{\Gamma, x : T \vdash t_1 : S}{\Gamma \vdash \lambda x : T.t_1 : T \to S} \quad \Gamma \vdash t_2 : T}{\Gamma \vdash (\lambda x : T.t_1)t_2 : S} \quad \rightsquigarrow \quad \dfrac{\dfrac{\overset{[T]}{\Pi_1}}{\dfrac{S}{T \supset S}} \qquad \overset{\Pi_2}{T}}{S}$$

Quello che ci dice l'albero di destra, è:

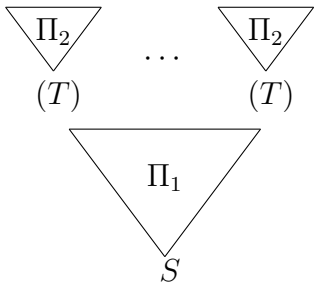
(a) Voglio dimostrare S;

(b) Dimostro che, supponendo T, vale S;

(c) Dimostro separatamente T;

(d) Da (b) e (c) concludo S.

L'applicazione della regola di β-riduzione alla dimostrazione produce l'albero:



cioè la dimostrazione di T viene replicata ovunque serva all'interno della dimostrazione di S. La computazione, sotto forma di β-riduzione, corrisponde all'operazione logica di "proof normalisation" della deduzione naturale.

Per ulteriori discussioni sulla corrispondenza di Curry–Howard, vedi Girard et al. [1989]; Gallier [1993]; Sørensen and Urzyczyn [2006].

2.3 Safety

Dimostriamo la proprietà di *safety* per il lambda calcolo tipizzato semplice in modo del tutto analogo a quanto fatto per il linguaggio delle espressioni aritmetiche e booleane. Al solito, ci servono alcuni lemmi e teoremi preliminari.

Lemma 2.5 (Inversione)**.**

$$\begin{array}{lll} \textit{se} & \Gamma \vdash x : R & \textit{allora} \quad x : R \in \Gamma \\ \textit{se} & \Gamma \vdash \lambda x : T.t : S & \textit{allora} \quad S \equiv T \to R \quad \textit{e} \quad \Gamma, x : T \vdash t : R \\ \textit{se} & \Gamma \vdash t_1 t_2 : S & \textit{allora} \quad \Gamma \vdash t_1 : T \to S \quad \textit{e} \quad \Gamma \vdash t_2 : T \end{array}$$

Dimostrazione. Immediata in tutti i casi, per ispezione delle regole. □

Teorema 2.6 (Unicità del tipo)**.** *Sia t un termine e $FV(t) \subseteq \Gamma$ (cioè il contesto Γ assegna un tipo a tutte le variabili libere in t):*

$$\textit{se} \quad \Gamma \vdash t : T \quad \textit{allora } T \textit{ è unico (e con derivazione unica)}$$

Dimostrazione. Per induzione sulla derivazione di $\Gamma \vdash t : T$ analogamente al caso delle espressioni aritmetiche e booleane. □

Lemma 2.7 (Forme canoniche)**.** *Sia v un valore. Allora:*

$$\begin{array}{lll} \textit{se} & \vdash v : \mathit{Bool} & \textit{allora} \quad v \equiv \mathtt{tt} \quad \textit{oppure} \quad v \equiv \mathtt{ff} \\ \ldots & & \textit{(analogamente per tutti gli altri tipi base)} \end{array}$$

$$\textit{se} \quad \vdash v : T \to S \quad \textit{allora} \quad v \equiv \lambda x : T.t$$

Dimostrazione. Di nuovo, immediata via il lemma di inversione. □

Osserviamo che nel teorema precedente, i giudizi sono nella forma:

$$\vdash t : T$$

Questo significa che t ha tipo T e che t è *chiuso* (t è tipizzabile senza contesto, cioè $FV(t) = \emptyset$). Questo non costituisce una limitazione, in quanto generalmente i programmi sono termini chiusi.

Teorema 2.8 (Progresso)**.**

$$\textit{se} \quad \vdash t : T \quad \textit{allora } t \textit{ è un valore oppure } t \to t' \textit{ per qualche } t'$$

Dimostrazione. Per induzione sulla derivazione di $\vdash t : T$.

- Casi base.
  - (i) $\vdash x : T$ impossibile senza contesto (vacuamente vero);
  - (ii) $t \equiv \lambda x : T_1.t_1$ e $T \equiv T_1 \to T_2$. Vero, in quanto t è un valore.

- Caso induttivo.

$$\frac{\vdash t_1 : S{\to}T \qquad \vdash t_2 : S}{\vdash t_1 t_2 : T}$$

  Per casi sulla premessa di sinistra. Per ipotesi induttiva t_1 è un valore oppure $t_1 \to t_1'$:

  - t_1 è un valore. Per casi sulla premessa di destra. Per ipotesi induttiva t_2 è un valore oppure $t_2 \to t_2'$:
    - t_2 è un valore. Per il lemma 2.7 (*forme canoniche*):

$$t_1 \equiv \lambda x : S.s$$

      perciò è applicabile la regola β:

$$t_1 t_2 \to s[t_2/x]$$

    - $t_2 \to t_2'$. Siccome siamo nel caso in cui t_1 è un valore, si può applicare la regola:

$$\frac{t_2 \to t_2'}{t_1 t_2 \to t_1 t_2'}$$

  - $t_1 \to t_1'$. Possiamo applicare un passo di riduzione:

$$\frac{t_1 \to t_1'}{t_1 t_2 \to t_1' t_2}$$

□

Teorema 2.9 (Preservazione o Subject reduction)**.**

$$\textit{se} \quad \Gamma \vdash t : T \quad e \quad t \to t' \quad \textit{allora} \quad \Gamma \vdash t' : T$$

Dimostrazione. Per induzione sulla derivazione di $\Gamma \vdash t : T$.

- Casi base.

  (i) $\Gamma \vdash x : T$. Vacuamente vero, in quanto $x \not\to$;

  (ii) $\Gamma \vdash t \equiv \lambda x : T_1.t_1 \ : T$. Di nuovo, è banalmente vero, poichè $t \not\to$.

- Caso induttivo.

$$\frac{\Gamma \vdash t_1 : S{\to}T \qquad \Gamma \vdash t_2 : S}{\Gamma \vdash t_1 t_2 : T}$$

Si procede per casi sulla derivazione di $t \to t'$:

– $\dfrac{t_1 \to t'_1}{t_1 t_2 \to t'_1 t_2}$

Sappiamo che:

a) $\Gamma \vdash t_2 : S$

b) $\Gamma \vdash t_1 : S{\to}T$ e $t_1 \to t'_1$, da cui, per ipotesi induttiva:

$$\Gamma \vdash t'_1 : S{\to}T$$

e pertanto:

$$\frac{\Gamma \vdash t'_1 : S{\to}T \qquad \Gamma \vdash t_2 : S}{\Gamma \vdash t'_1 t_2 : T}$$

– Analogo il caso $\dfrac{t_2 \to t'_2}{t_1 t_2 \to t_1 t'_2}$ con t_1 valore;

– $t \equiv t_1 t_2 \equiv (\lambda x : S.s) t_2$ con t_2 valore e:

$$t \ \to \ t' \equiv s[t_2/x]$$

Adesso abbiamo le seguenti informazioni:

a) $\Gamma \vdash t_2 : S$;

b) $\dfrac{\Gamma, x : S \vdash s : T}{\Gamma \vdash \lambda x : S.s : S{\to}T}$

Tuttavia, per concludere questo caso, e quindi la dimostrazione, dovremmo poter affermare che il termine $s[t_2/x]$ *perserva il tipo* T, o più in generale, che il tipo di un termine è preservato dall'operazione di sostituzione. Questo risultato ci è fornito dal lemma di *preservazione sotto sostituzione*, che dimostreremo ora, e da cui concludiamo.

□

Lemma 2.10 (Preservazione sotto sostituzione)**.**

$$\textit{se} \quad \Gamma, x : S \vdash r : R \quad \textit{e} \quad \Gamma \vdash s : S \quad \textit{allora} \quad \Gamma \vdash r[s/x] : R$$

Dimostrazione. Per induzione sulla struttura di r. Sia y una variabile t.c. $y \not\equiv x$.

(i) $r \equiv x$, quindi $\Gamma, x : S \vdash x : R$

$$r[s/x] \equiv x[s/x] \equiv s \quad \text{per ipotesi} \quad \Gamma \vdash s : S \equiv R$$

(ii) $r \equiv y$, quindi $\Gamma, x : S \vdash y : R$

$$r[s/x] \equiv y[s/x] \equiv y \quad \text{e} \quad \Gamma \vdash y : R$$

(iii) $r \equiv \lambda y : T_1.t_2$, quindi:

$$\frac{\Gamma, x : S, y : T_1 \vdash t_2 : T_2}{\Gamma, x : S \vdash \lambda y : T_1.t_2 : T_1 \to T_2}$$

da cui, per ipotesi induttiva:

$$\frac{\Gamma, y : T_1 \vdash t_2[s/x] : T_2}{\Gamma \vdash \lambda y : T_1.t_2[s/x] : T_1 \to T_2}$$

perciò possiamo banalmente concludere:

$$\Gamma \vdash (\lambda y : T_1.t_2)[s/x] : T_1 \to T_2 \equiv R$$

(iv) $r \equiv \lambda x : T_1.t_2$, allora $(\lambda x : T_1.t_2)[s/x] \equiv \lambda x : T_1.t_2 \equiv r$.

(v) $r \equiv t_1 t_2$. Per il lemma di inversione $\Gamma, x : S \vdash t_1 : T \to R$ e $\Gamma, x : S \vdash t_2 : T$. Quindi, per ipotesi induttiva applicata al sottotermine t_1, otteniamo $\Gamma \vdash t_1[s/x] : T \to R$, mentre applicandola su t_2 otteniamo $\Gamma \vdash t_2[s/x] : T$. Ora:

$$\frac{\Gamma \vdash t_1[s/x] : T \to R \qquad \Gamma \vdash t_2[s/x] : T}{\Gamma \vdash (t_1[s/x]\ t_2[s/x]) : R}$$

e ovviamente $(t_1[s/x]\ t_2[s/x]) \equiv (t_1\ t_2)[s/x] \equiv r[s/x]$.

□

2.4 Cancellazione del tipo

Notiamo che le annotazioni di tipo non giocano alcun ruolo durante la valutazione dei termini (non facciamo nessun controllo sui tipi a run–time). Tutto ciò che facciamo è semplicemente "riportare" le annotazioni di tipo durante la valutazione.

La maggior parte dei compilatori per linguaggi di programmazione completi, non riportano le annotazioni durante la valutazione: queste sono usate solamente nella fase di typechecking (e nella generazione del codice, nei compilatori più avanzati) ma non appaiono nel codice oggetto (i.e. il programma compilato). In effetti, i programmi vengono riconvertiti in una forma non tipizzata prima di essere eseguiti. Questo stile di semantica può essere formalizzato usando una *funzione di cancellazione* che mappa i termini tipizzati nei corrispettivi senza tipi:

Definizione 2.11 (Funzione di cancellazione o erasure)**.**

$$\begin{aligned} er(x) &\triangleq x \\ er(t_1 t_2) &\triangleq er(t_1)\, er(t_2) \\ er(\lambda x : T.t) &\triangleq \lambda x.er(t) \end{aligned}$$

Naturalmente, ci aspettiamo che i due modi di fornire la semantica del calcolo coincidano: sarà equivalente valutare un termine tipizzato direttamente, o valutare (con la classica strategia weak call-by-value) il termine corrispettivo senza tipi. Questa proprietà è catturata dal seguente teorema, che si può riassumere nella frase "la valutazione commuta con la cancellazione", cioè queste operazioni possono essere effettuate in qualunque ordine: raggiungiamo lo stesso termine sia valutando ed effettuando la cancellazione, sia cancellando i tipi e poi valutando:

Teorema 2.12. *Sia t un termine tipizzato:*

1. *se $t \to t'$, allora $er(t) \to er(t')$ (con valutazione non tipizzata);*
2. *se $er(t) \to m'$ (con valutazione non tipizzata), allora esiste un termine tipizzato t' tale che $t \to t'$ e $er(t') = m'$.*

Dimostrazione. Immediata, per induzione sulle derivazioni delle valutazioni. □

Il problema dell'*inferenza di tipo* o di *ricostruzione del tipo*, chiede invece se, dato un termine non tipizzato t, esiste un modo per decorarlo con delle informazioni di tipo, ottenendo un termine t' che sia ben tipizzato e tale che $er(t') = t$.

Ovviamente, non esiste sempre un modo per decorare termini non tipizzati (si pensi a $\lambda x.xx$). Inoltre, come visto con le varianti tipizzate dei numerali di Church, il tipo di un termine non è unico. Un altro problema interessante potrebbe essere quello di determinare la forma più generale di tipo per un termine non tipizzato.

2.5 Normalizzazione

La *normalizzazione* è una proprietà del calcolo che dice che un termine (nel nostro caso, tipizzabile) valuta sempre ad una forma normale. Ne esistono almeno due differenti formulazioni:

- *Debole normalizzazione* o *weak normalization* (abbr. *WN*). Ogni termine può raggiungere una forma normale. Formalmente, per ogni termine t, esiste s forma normale tale che $t \to^* s$;
- *Forte normalizzazione* o *strong normalization* (abbr. *SN*). Ogni termine raggiunge sempre una forma normale. In altre parole, per ogni termine t, ogni successione di riduzioni che parte da t è finita.

Ovviamente (SN)⇒(WN). Queste due nozioni hanno senso solo in presenza di nondeterminismo nelle regole di β–riduzione, mentre coincidono negli altri casi (e nel nostro caso, weak call-by-value), dove per ogni termine t esiste una sola sequenza di riduzioni che parte da t.

Teorema 2.13. *(SN) Per ogni termine t del lambda calcolo tipizzato semplice:*

$$\textit{se} \quad \vdash t : T \quad \textit{allora} \quad \textit{esiste } v \textit{ valore t.c. } t \to^* v$$

Il primo approccio che viene in mente per dimostrare questo teorema è andare per induzione sulla derivazione di $\vdash t : T$, mostrando che c'è una quantità (la *dimensione* o *peso* di t) che decresce ad ogni passo di riduzione, analogamente a quanto fatto per il linguaggio delle espressioni aritmetiche e booleane. Questo approccio tuttavia non funziona se prendiamo come dimensione la semplice lunghezza del termine t: infatti nel caso dell'applicazione, vorremmo dimostrare che $t_1 t_2$ è normalizzante e per ipotesi induttiva avremmo t_1 e t_2 normalizzanti; siano v_1 e v_2 le loro rispettive forme normali. Per il lemma 2.5 (*inversione*), si avrebbe v_1 di tipo $T_{11} \to T_{12}$ per qualche T_{11} e T_{12} e, per il lemma 2.7 (*forme canoniche*), v_1 dovrebbe avere la forma $\lambda x : T_{11}.t_{12}$. Ma a questo punto, ridurre $t_1 t_2$ a $(\lambda x : T_{11}.t_{12})v_2$ non ci darebbe una forma normale, bensì il termine $t_{12}[v_2/x]$, e per concludere la dimostrazione dovremmo provare che questo termine è a sua volta normalizzante. Ma

questo non è vero in generale, poichè $t_{12}[v_2/x]$ può essere più grande di $t_1 t_2$ (la sostituzione può infatti produrre più copie di v_2 in t_{12}).

2.5.1 Normalizzazione debole

Consideriamo la versione completa (full) della β-riduzione, cioè la regola:

$$\begin{array}{ccccc} (\lambda x : T.s) & t & \to & s[t/x] \\ \cdot\cdot & \cdot\cdot & & \cdot\cdot \\ T{\to}S & T & & S \end{array}$$

chiusa *per ogni contesto*. Questa, a differenza di quella usata finora, è non-deterministica: il redex da ridurre ad ogni passo può essere scelto arbitrariamente.

Osserviamo che la β-riduzione genera termini che *non sono sottotermini dei termini originali*. Inoltre la dimensione del termine generato non è in generale in relazione con la dimensione del termine originale. Il nostro scopo è individuare una quantità che decresce strettamente ad ogni passo di riduzione. Osserviamo i modi in cui nuovi redex vengono a crearsi nel ridotto $s[t/x]$:

1. sono i residui dei redex già presenti in s e t;
2. quando $t \equiv \lambda y. \ldots$ e x compare dentro s in posizione funzionale (i.e. a primo membro di un'applicazione);
3. è un redex creato tra il ridotto ed il suo contesto. Ad esempio:

$$((\lambda u : T{\to}T.\lambda v : T.uv)x)y \to (\lambda v : T.xv)y$$

Consideriamo ora i tipi dei redex creati a seguito di una β-riduzione (il *tipo di un redex* è il tipo della sua parte funzione). Nel caso 2, il nuovo redex sarà di tipo T. Nel caso 3, il nuovo redex avrà tipo S. Nulla si può dire sul primo caso.

Definizione 2.14. *La* dimensione *di un tipo è definita da:*

$$\begin{array}{rcl} |o| & \triangleq & 1 \\ |T{\to}S| & \triangleq & 1 + |T| + |S| \end{array}$$

Definizione 2.15. *La* dimensione di un redex *è la dimensione del suo tipo (cioè la dimensione del tipo della sua parte funzione).*

Usiamo il nondeterminismo insito nella β-riduzione per imporre una strategia di valutazione che ci eviti sistematicamente di incorrere nel caso banale in cui il grado di un redex è uguale al grado del suo ridotto.

Definizione 2.16 (Strategia di normalizzazione)**.** *Ad ogni passo, scegli di ridurre un redex:*

- *di grado massimo;*
- *che non contenga al suo interno alcun altro redex di grado massimo.*

Osserviamo che se t è un termine e $t \to t'$ secondo questa strategia, il numero di redex di grado massimo di t' è *strettamente minore* di quello di t.

Definizione 2.17 (Peso di un termine)**.** *Sia t un termine. Il suo* peso $w(t)$ *è una coppia (n, m) dove:*

- *n è il numero di redex di grado massimo in t;*
- *m è il grado massimo dei redex in t.*

Ora possiamo affermare che, se $t \to t'$ secondo la strategia di normalizzazione, allora:

$$w(t) > w(t')$$

dove "$>$" è l'operazione di *ordinamento lessicografico tra coppie*. Da qui, otteniamo la *normalizzazzione debole* per la full β-reduction nel lambda calcolo tipizzato semplice.

2.5.2 Normalizzazione forte

Indichiamo con $\eta(t)$ la proprietà di forte normalizzazione per il termine tipizzato t. Per t chiuso, definiamo degli insiemi indicizzati per tipo, che definiscono delle proprietà (o *relazioni logiche*) sui termini:

$$\begin{array}{rlll} R_o(t) & \text{sse} & \eta(t) & \\ R_{T_1 \to T_2}(t) & \text{sse} & \eta(t) & \text{e} \quad \text{per ogni termine } s, \text{ se } R_{T_1}(s) \text{ allora } R_{T_2}(ts) \end{array}$$

La dimostrazione di forte normalizzazione del calcolo è divisa in due parti:

(i) se $R_T(t)$ allora $\eta(t)$. Ovvio: per definizione, ogni termine che soddisfa la proprietà R_T, è fortemente normalizzante;

(ii) per ogni t, se $\vdash t : T$ allora $R_T(t)$. Per dimostrare questa seconda proprietà, abbiamo bisogno di provare due ulteriori lemmi: il primo ci dirà che la proprietà R_T è preservata dalla riduzione e dall'espansione. La seconda che tutti i termini di tipo T soddisfano R_T.

Lemma 2.18 (Chiusura di R_T per riduzione ed espansione)**.** *Sia t un termine tale che $\vdash t : T$ e $t \to t'$. Allora:*

$$R_T(t) \quad sse \quad R_T(t')$$

Dimostrazione. Innanzitutto osserviamo che $\eta(t)$ sse $\eta(t')$ (cioè se t raggiunge sempre una forma normale, allora anche dopo un passo di riduzione otteniamo un termine t' che è normalizzante, e viceversa). Procediamo per induzione sulla struttura di T:

- Caso base: T è di tipo base ($T \equiv o$). Ovvio: per definizione di R_o, otteniamo $\eta(t)$ sse $\eta(t')$, che vale sempre, come abbiamo già osservato.
- Caso induttivo: $T \equiv T_1 \to T_2$. Dimostriamo separatamente i due versi del "se e solo se":
  - se $R_{T_1 \to T_2}(t)$ allora $R_{T_1 \to T_2}(t')$, cioè:
    - $\eta(t')$ come già osservato, ovvio;
    - per ogni termine s, se $R_{T_1}(s)$ allora $R_{T_2}(t's)$. Sappiamo che:
      - se $R_{T_1}(s)$ allora $R_{T_2}(ts)$ per ipotesi;
      - $\dfrac{t \to t'}{ts \to t's}$

      e da questi due fatti, per ipotesi induttiva, concludiamo il primo verso della doppia implicazione;
  - se $R_{T_1 \to T_2}(t')$ allora $R_{T_1 \to T_2}(t)$. L'argomento è del tutto simmetrico al caso precedente.

□

Lemma 2.19 (Caratterizzazione di R_T)**.** *Sia t un termine tale che $x_1 : T_1, \ldots, x_n : T_n \vdash t : T$ e siano $v_1, \ldots, v_n$ valori chiusi tali che $\vdash v_i : T_i$ e $R_{T_i}(v_i)$ per ogni $1 \leq i \leq n$. Allora:*

$$R_T(t[v_1/x_1, \ldots, v_n/x_n])$$

Dimostrazione. Per induzione sulla derivazione di $\{x_i : T_i\}_{i=1,\ldots,n} \vdash t : T$.

- Caso base: $t \equiv x_i$ per qualche $1 \leq i \leq n$.

$$\frac{}{x_1 : T_1, \ldots, x_n : T_n \vdash x_i : T_i}$$

$R_{T_i}(x_i[v_1/x_1, \ldots, v_n/x_n])$ sse $R_{T_i}(v_i)$, valida per ipotesi.

- Casi induttivi.
  - Applicazione: $t \equiv t_1 t_2$.

$$\frac{\{x_i : T_i\}_{i=1,\ldots,n} \vdash t_1 : S \to T \qquad \{x_i : T_i\}_{i=1,\ldots,n} \vdash t_2 : S}{\{x_i : T_i\}_{i=1,\ldots,n} \vdash t_1 t_2 : T}$$

    Ora abbiamo:
    - $R_{S \to T}(t_1[v_1/x_1, \ldots, v_n/x_n])$ *(per ipotesi induttiva sulla premessa di sinistra)*;
    - $R_S(t_2[v_1/x_1, \ldots, v_n/x_n])$ *(per ipotesi induttiva sulla premessa di destra)*;
    - $R_T(t_1[v_1/x_1, \ldots, v_n/x_n] t_2[v_1/x_1, \ldots, v_n/x_n])$ *(dalle due precedenti, per definizione di $R_{S \to T}$).*

    da cui, per definizione di sostituzione:

$$R_T((t_1 t_2)[v_1/x_1, \ldots, v_n/x_n])$$

  - Astrazione: $t \equiv \lambda y : S_1.t_2$.

$$\frac{x_1 : T_1, \ldots, x_n : T_n, y : S_1 \vdash t_2 : S_2}{\{x_i : T_i\}_{i=1,\ldots,n} \vdash \lambda y : S_1.t_2 : S_1 \to S_2}$$

    Ovviamente, vale $\eta((\lambda y : S_1.t_2)[v_1/x_1, \ldots, v_n/x_n])$. Resta da dimostrare che, per ogni termine s:

$$\text{se} \quad R_{S_1}(s) \quad \text{allora} \quad R_{S_2}((\lambda y : S_1.t_2)[v_1/x_1, \ldots, v_n/x_n]\ s)$$

    Se vale $R_{S_1}(s)$, allora vale anche $\eta(s)$, cioè esiste un certo valore v tale che $s \to^* v$. Per il lemma 2.18 (R_{S_1} *è chiuso per riduzione*) applicato ad ogni passo di riduzione, sappiamo che vale anche $R_{S_1}(v)$. Per ipotesi induttiva, abbiamo:

$$R_{S_2}(t_2[v_1/x_1, \ldots, v_n/x_n, v/y])$$

    e di nuovo, per il lemma 2.18 (stavolta applicato nel senso contrario, ossia R_{S_2} *è chiuso per espansione*), otteniamo:

$$R_{S_2}((\lambda y : S_1.t_2)[v_1/x_1, \ldots, v_n/x_n]\ s)$$

□

Concludiamo osservando che, per $n = 0$, il lemma precedente diventa:

$$\text{se} \quad \vdash t : T \quad \text{allora} \quad R_T(t)$$

e quindi $\eta(t)$, cioè ogni termine ben tipizzato e chiuso normalizza, che è la proprietà di forte normalizzazione del calcolo cui eravamo interessati.

2.6 Espressività del calcolo

Indichiamo con:

$$\begin{array}{ll} \lambda\beta & \text{il } \lambda\text{-calcolo puro (senza tipi)} \\ \lambda\beta^T & \text{il } \lambda\text{-calcolo tipizzato semplice} \end{array}$$

Teorema 2.20 ($\lambda\beta$ è Turing–completo)**.** *Sia $f \in \mathcal{C}$ una funzione calcolabile. Allora esiste M_f λ-termine tale che, per ogni m, n naturali:*

$$\begin{array}{llll} \textit{se} & f(n) \simeq m & \textit{allora} & M_f\underline{n} \to^* \underline{m} \\ \textit{se} & f(n) \uparrow & \textit{allora} & M_f\underline{n} \textit{ non ha f.n.} \end{array}$$

Per contro, è abbastanza evidente che $\lambda\beta^T$ rappresenta solo funzioni totali. Infatti:

$$Y \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

non è tipizzabile, banalmente per via dell'autoapplicazione. È interessante notare come la *forte normalizzazione* ci permette di decidere proprietà generalmente indecidibili per il lambda calcolo senza tipi, come la β-equivalenza fra termini.

Definizione 2.21 (Definibilità)**.** *$f : \mathbb{N}^k \to \mathbb{N}$ è (strettamente) definibile in $\lambda\beta^T$ sse esiste un termine F tale che:*

$$\vdash F : \underbrace{N_o \to N_o \to \cdots \to N_o}_{k\text{-volte}} \to N_o$$

(dove $N_o \equiv (o \to o) \to o \to o$ è il tipo delle varianti tipizzate dei numerali di Church) e, per ogni $n_1, \ldots, n_k, m \in \mathbb{N}$, vale:

$$f(n_1, \ldots, n_k) = m \quad \textit{sse} \quad F\underline{n}_1^T \cdots \underline{n}_k^T \to^* \underline{m}$$

Definizione 2.22 (Polinomi estesi)**.** *La* classe dei polinomi estesi *$\mathcal{P}$ è la più piccola classe di funzioni $f : \mathbb{N}^k \to \mathbb{N}$ che contiene:*

- *le costanti* `0` *e* `1`*;*
- *le proiezioni $\pi_i^k : \mathbb{N}^k \to \mathbb{N}$;*
- *le funzioni di somma e di moltiplicazione;*
- *la funzione* `if-then-else`*;*

e chiusa per composizione.

Teorema 2.23 (Schwichtenberg)**.** *f è (strettamente) definibile sse $f \in \mathcal{P}$.*

Questa è la formulazione classica, che insiste sull'uniformità sia del tipo del termine F che codifica la funzione, sia dei numerali. Si possono dare altre nozioni di definibilità, meno stringenti – vuoi nei tipi permessi per F, vuoi nei termini (e nei relativi tipi) scelti per rappresentare i numeri naturali – che permettono di ampliare la classe delle funzioni definibili (per una trattazione formale, vedi Sørensen and Urzyczyn [2006]).

2.7 Ricostruzione del tipo

Definizione 2.24 (Sostituzione di tipi)**.** *Una* sostituzione di tipi *è una funzione σ da un dominio finito di* variabili di tipo *in tipi.*

$$\sigma \quad : \quad \texttt{VarTipo}\to\texttt{Tipi}$$

$$\begin{array}{rcl} \sigma(X) & \triangleq & \begin{cases} T & \textit{se } X\mapsto T \in \sigma \\ X & \textit{altrimenti} \end{cases} \\ \sigma(o) & \triangleq & o \quad \textit{(per ogni } o \textit{ tipo base)} \\ \sigma(T_1\to T_2) & \triangleq & \sigma(T_1)\to\sigma(T_2) \end{array}$$

Per $\Gamma = \{x_1 : T_1, \ldots, x_n : T_n\}$ contesto, la sostituzione di tipi diventa:

$$\sigma(\Gamma) \triangleq \{x_1 : \sigma(T_1), \ldots, x_n : \sigma(T_n)\}$$

Definizione 2.25 (Composizione di sostituzioni)**.** *Siano σ, τ sostituzioni:*

$$\sigma \circ \tau \triangleq \begin{cases} X\mapsto\sigma(T) & \textit{se} \quad X\mapsto T \in \tau \\ X\mapsto T & \textit{se} \quad X\mapsto T \in \sigma \quad \textit{e} \quad X \notin \textit{dom}(\tau) \end{cases}$$

Osserviamo che, al solito:

$$(\sigma \circ \tau)(T) \equiv \sigma(\tau(T))$$

Un'altra notazione con cui si possono indicare le sostituzioni di tipi la composizione di sostituzioni è:

$$\begin{array}{ccc} T\sigma & \textit{per} & \sigma(T) \\ \sigma\tau & \textit{per} & \tau \circ \sigma \end{array}$$

Definizione 2.26 (Sostituzione su termini)**.** *Siano t un termine e σ una sostituzione di tipi. $\sigma(t)$ è definita da:*

$$\begin{aligned} \sigma(x) &\triangleq x \\ \sigma(t_1 t_2) &\triangleq \sigma(t_1)\,\sigma(t_2) \\ \sigma(\lambda x : T.t_1) &\triangleq \lambda x : \sigma(T).\sigma(t_1) \end{aligned}$$

Osserviamo la seguente proprietà: siano σ una sostituzione e sia $\Gamma \vdash t : T$. Allora:

$$\sigma(\Gamma) \vdash \sigma(t) : \sigma(T)$$

Dimostrazione. Per induzione sulla derivazione di $\Gamma \vdash t : T$.

- Caso base.

  $$\textit{se} \quad \frac{x : T \in \Gamma}{\Gamma \vdash x : T} \qquad \textit{allora} \quad \frac{x : \sigma(T) \in \sigma(\Gamma)}{\sigma(\Gamma) \vdash x : \sigma(T)}$$

  per definizione di $\sigma(\Gamma)$, e per la definizione 2.26 (*sostituzione applicata a termini*) $\sigma(\Gamma) \vdash \sigma(x) : \sigma(T)$.

- Casi induttivi:

  - applicazione.

    $$\frac{\Gamma \vdash t_1 : S \to T \qquad \Gamma \vdash t_2 : S}{\Gamma \vdash t_1 t_2 : T}$$

    Ora, per ipotesi induttiva sulla premessa di sinistra e per definizione di σ:

    $$\sigma(\Gamma) \vdash \sigma(t_1) : \sigma(S) \to \sigma(T)$$

    mentre per ipotesi induttiva sulla premessa di destra:

    $$\sigma(\Gamma) \vdash \sigma(t_2) : \sigma(S)$$

    da cui:

    $$\frac{\sigma(\Gamma) \vdash \sigma(t_1) : \sigma(S) \to \sigma(T) \qquad \sigma(\Gamma) \vdash \sigma(t_2) : \sigma(S)}{\sigma(\Gamma) \vdash \sigma(t_1)\sigma(t_2) : \sigma(T)}$$

    e, per definizione di $\sigma(t_1 t_2)$:

    $$\sigma(\Gamma) \vdash \sigma(t_1 t_2) : \sigma(T)$$

  - λ-astrazione.

$$\frac{\Gamma, x : S \vdash t_2 : T}{\Gamma \vdash \lambda x : S.t_2 : T}$$

Per ipotesi induttiva:

$$\frac{\sigma(\Gamma), x : \sigma(S) \vdash \sigma(t_2) : \sigma(T)}{\sigma(\Gamma) \vdash \lambda x : \sigma(S).\sigma(t_2) : \sigma(T)}$$

e, per definizione di $\sigma(\lambda x : S.t_2)$:

$$\sigma(\Gamma) \vdash \sigma(\lambda x : S.t_2) : \sigma(T)$$

□

Ora è possibile formulare il *problema di ricostruzione del tipo* più precisamente: dati un contesto Γ ed un termine t, determinare una sostituzione σ ed un tipo T tali che:

$$\sigma(\Gamma) \vdash \sigma(t) : T$$

La soluzione è suddivisa in due fasi:

1. *Generare vincoli sui tipi*, dove un vincolo è un'equazione della forma:

   $$T = S$$

   con T ed S tipi. In questa fase generiamo un *insieme di vincoli* della forma:

   $$\mathcal{C} = \{T_i = S_i | i = 1, \ldots, n\}$$

2. *Risolvere l'insieme di vincoli*, cioè determinare una sostituzione σ (se esiste) che soddisfa $\mathcal{C}$, cioè tale che:

   $$\sigma(T_i) = \sigma(S_i)$$

   per ogni vincolo $T_i = S_i$ contenuto in $\mathcal{C}$.

Definizione 2.27. *Definiamo per induzione una nozione di* giudizio di tipo con vincoli*, nella forma* $\Gamma \vdash t : T \mid_\chi \mathcal{C}$ *(il simbolo* χ *a pedice indica l'insieme di variabili di tipo che serve a tener traccia delle nuove variabili introdotte ad ogni passo di derivazione):*

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T \mid_\varnothing \{\}}\ (\text{CT}_{\text{VAR}}) \qquad \frac{\Gamma, x : T \vdash t : S \mid_\chi \mathcal{C}}{\Gamma \vdash \lambda x : T.t : T \to S \mid_\chi \mathcal{C}}\ (\text{CT}_{\text{ABS}})$$

$$\frac{\Gamma \vdash t_1 : T_1 \mid_{\chi'} \mathcal{C}_1 \qquad \Gamma \vdash t_2 : T_2 \mid_{\chi''} \mathcal{C}_2}{\Gamma \vdash t_1 t_2 : X \mid_{\chi' \cup \chi'' \cup \{X\}} \mathcal{C}_1 \cup \mathcal{C}_2 \cup \{T_1 = T_2 \to X\}} \quad \begin{array}{l} \textit{purchè } X \textit{ sia fresca e} \\ \chi' \cap \chi'' = \chi' \cap FV(T_2) = \\ = \chi'' \cap FV(T_1) = \varnothing \end{array} \quad (\text{CT}_{\text{APP}})$$

$$\Gamma \vdash \texttt{0} : \texttt{Nat} \mid_{\varnothing} \{\} \quad (\text{CT}_{\text{ZERO}}) \qquad \frac{\Gamma \vdash t : T \mid_{\chi} \mathcal{C}}{\Gamma \vdash \texttt{succ}\ t : \texttt{Nat} \mid_{\chi} \mathcal{C} \cup \{T = \texttt{Nat}\}}\ (\text{CT}_{\text{SUCC}})$$

$$\Gamma \vdash \texttt{tt} : \texttt{Bool} \mid_{\varnothing} \{\} \quad (\text{CT}_{\text{TT}}) \qquad \frac{\Gamma \vdash t : T \mid_{\chi} \mathcal{C}}{\Gamma \vdash \texttt{pred}\ t : \texttt{Nat} \mid_{\chi} \mathcal{C} \cup \{T = \texttt{Nat}\}}\ (\text{CT}_{\text{PRED}})$$

$$\Gamma \vdash \texttt{ff} : \texttt{Bool} \mid_{\varnothing} \{\} \quad (\text{CT}_{\text{FF}}) \qquad \frac{\Gamma \vdash t : T \mid_{\chi} \mathcal{C}}{\Gamma \vdash \texttt{iszero}\ t : \texttt{Bool} \mid_{\chi} \mathcal{C} \cup \{T = \texttt{Nat}\}}\ (\text{CT}_{\text{ISZERO}})$$

$$\frac{\Gamma \vdash t_1 : T_1 \mid_{\chi'} \mathcal{C}_1 \qquad \Gamma \vdash t_2 : T_2 \mid_{\chi''} \mathcal{C}_2 \qquad \Gamma \vdash t_3 : T_3 \mid_{\chi'''} \mathcal{C}_3}{\Gamma \vdash \texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T_2 \mid_{\chi' \cup \chi'' \cup \chi'''} \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3 \cup \{T_1 = \texttt{Bool}, T_2 = T_3\}}\ (\text{CT}_{\text{IF}})$$

Anche nel caso della regola (CT$_{\text{IF}}$) bisognerebbe esplicitare ulteriori condizioni a margine sugli insiemi di variabili utilizzati. Per una formalizzazione più completa di queste regole, vedi Pierce [2002].

Definizione 2.28. *La sostituzione* σ soddisfa *(o* verifica *o* è un unificatore per*)* un insieme di vincoli $\mathcal{C}$ *sse:*

$$\textit{per ogni} \quad T_i = S_i \in \mathcal{C} \quad \textit{si ha} \quad \sigma(T_i) \equiv \sigma(S_i)$$

Definizione 2.29. *Una* soluzione *per il problema di ricostruzione del tipo* $(\Gamma\ ;\ t\ ;\ S\ ;\ \mathcal{C})$ *è una coppia* (σ, T) *tale che:*

- σ *soddisfa* $\mathcal{C}$*;*
- $\sigma(S) \equiv T$

Le regole di tipizzazione con vincoli definite in (2.27) costituiscono un algoritmo induttivo che permette di trovare una soluzione al problema di ricostruzione del tipo. Verifichiamo che tale algoritmo è corretto e completo rispetto alla nozione di *soluzione al problema di ricostruzione*.

Teorema 2.30 (Correttezza dell'algoritmo con vincoli)**.** *Sia* $\Gamma \vdash t : S \mid_{\chi} \mathcal{C}$ *e sia* (σ, T) *una soluzione per* $(\Gamma\ ;\ t\ ;\ S\ ;\ \mathcal{C})$*. Allora:*

$$\sigma(\Gamma) \vdash \sigma(t) : T$$

Dimostrazione. Per induzione sulla derivazione di $\Gamma \vdash t : S \mid \mathcal{C}$ (omettiamo χ perchè non gioca alcun ruolo nella dimostrazione):

- Casi base.
  - CT$_{\text{VAR}}$:

$$\frac{x : S \in \Gamma}{\Gamma \vdash x : S \mid \{\}}$$

Sia (σ, T) soluzione, cioè $\sigma(S) \equiv T$. Immediato. $x : \sigma(S) \in \Gamma$, cioè $x : T \in \Gamma$, da cui:

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T}$$

– CTzero, CTtt, CTff banali, per definizione di sostituzione.

- Casi induttivi.

  – Astrazione. Sia (σ, T) soluzione per $(\Gamma\ ;\ \lambda x : T_1.t_2\ ;\ T_1 \to T_2\ ;\ \mathcal{C})$, cioè:

  (i) σ soddisfa $\mathcal{C}$;
  (ii) $\sigma(T_1 \to T_2) \equiv T \equiv \sigma(T_1) \to \sigma(T_2)$.

  Allora $(\sigma, \sigma(T_2))$ è soluzione di $(\Gamma, x : T_1\ ;\ t_2\ ;\ T_2\ ;\ \mathcal{C})$. Per ipotesi induttiva, si ha:

  $$\sigma(\Gamma, x : T_1) \vdash \sigma(t_2) : \sigma(T_2)$$

  e per definizione di *sostituzione applicata ai contesti*:

  $$\frac{\sigma(\Gamma),\ x : \sigma(T_1) \vdash \sigma(t_2) : \sigma(T_2)}{\sigma(\Gamma) \vdash \lambda x : \sigma(T_1).\sigma(t_2) : \sigma(T_1 \to T_2)}$$

  cioè, per (2.26) e per (ii):

  $$\sigma(\Gamma) \vdash \sigma(\lambda x : T_1.t_2) : T$$

  – Applicazione. Sia (σ, T) soluzione per:

  $$(\Gamma\ ;\ t_1 t_2\ ;\ X\ ;\ \mathcal{C}_1 \cup \mathcal{C}_2 \cup \{T_1 = T_2 \to X\})$$

  cioè:

  (i) σ soddisfa $\mathcal{C}_1$;
  (ii) σ soddisfa $\mathcal{C}_2$;
  (iii) $\sigma(T_1) \equiv \sigma(T_2) \to \sigma(X)$.
  (iv) $\sigma(X) \equiv T$

  Allora $(\sigma, \sigma(T_1))$ è soluzione di $(\Gamma\ ;\ t_1\ ;\ T_1\ ;\ \mathcal{C}_1)$ mentre $(\sigma, \sigma(T_2))$ è soluzione di $(\Gamma\ ;\ t_2\ ;\ T_2\ ;\ \mathcal{C}_2)$. Per ipotesi induttiva e per (iii), abbiamo quindi:

  $$\frac{\sigma(\Gamma) \vdash \sigma(t_1) : \sigma(T_2) \to \sigma(X) \qquad \sigma(\Gamma) \vdash \sigma(t_2) : \sigma(T_2)}{\sigma(\Gamma) \vdash \sigma(t_1)\sigma(t_2) : \sigma(X)}$$

cioè, per (2.26) e per (iv):

$$\sigma(\Gamma) \vdash \sigma(t_1 t_2) : T$$

– Gli altri casi procedono analogamente, per induzione diretta.

□

Teorema 2.31 (Completezza)**.** *Sia* $\Gamma \vdash t : S \mid_\chi \mathcal{C}$ *e sia* (σ, T) *tale che* $\sigma(\Gamma) \vdash \sigma(t) : T$ *e* $\mathrm{dom}(\sigma) \cap \chi = \varnothing$*. Allora esiste una sostituzione* σ' *tale che* (σ', T) *è soluzione di* $(\Gamma\,;\, t\,;\, S\,;\, \mathcal{C})$ *e* σ' *coincide con* σ *eccetto che sulle variabili in* χ*.*

Dimostrazione. Sia $\Gamma \vdash t : S \mid_\chi \mathcal{C}$ e siano (σ, T) tali che $\sigma(\Gamma) \vdash \sigma(t) : T$ con $\mathrm{dom}(\sigma) \cap \chi = \varnothing$. Procediamo per induzione sulla derivazione di $\Gamma \vdash t : S \mid_\chi \mathcal{C}$.

- Casi base.
  - CTVAR:
    $$\frac{x : S \in \Gamma}{\Gamma \vdash x : S \mid_\varnothing \{\}}$$
    Siano (σ, T) tali che $\sigma(\Gamma) \vdash \sigma(t) : T$. Allora:
    (i) σ soddisfa i vincoli (vacuamente);
    (ii) $\sigma(\Gamma) \vdash \sigma(t) : T$, da cui per inversione, $x : T \in \sigma(\Gamma)$, e quindi $\sigma(S) \equiv T$.
  - I casi per i tipi primitivi sono ovvi.
- Casi induttivi.
  - Astrazione.
    $$\frac{\Gamma, x : T_1 \vdash t_2 : T_2 \mid_\chi \mathcal{C}}{\Gamma \vdash \lambda x : T_1 . t_2 : T_1 \to T_2 \mid_\chi \mathcal{C}}$$
    Sia $\sigma(\Gamma) \vdash \lambda x : \sigma(T_1).\sigma(t_2) : T$. Per il lemma d'inversione:
    (i) $T \equiv \sigma(T_1) \to S_2$;
    (ii) $\sigma(\Gamma), x : \sigma(T_1) \vdash \sigma(t_2) : S_2$;
    Per ipotesi induttiva esistono (σ', S_2) soluzione per $(\Gamma, x : T_1\,;\, t_2\,;\, T_2\,;\, \mathcal{C})$ e σ, σ' coincidono a meno che sulle variabili in χ. Da qui:
    (iii) σ, σ' soddisfano $\mathcal{C}$;
    (iv) $T \equiv \sigma(T_1) \to S_2 \equiv \sigma'(T_1) \to S_2 \equiv \sigma'(T_1) \to \sigma'(T_2) \equiv \sigma'(T_1 \to T_2) \equiv \sigma'(S)$.

– Applicazione.

$$\frac{\Gamma \vdash t_1 : S_1 \mid_{\chi'} \mathcal{C}_1 \qquad \Gamma \vdash t_2 : S_2 \mid_{\chi''} \mathcal{C}_2}{\Gamma \vdash t_1 t_2 : X \mid_{\chi' \cup \chi'' \cup \{X\}} \mathcal{C}_1 \cup \mathcal{C}_2 \cup \{S_1 = S_2 \to X\}}$$

Inoltre X è fresca e $\chi' \cap \chi'' = \chi' \cap FV(T_2) = \chi'' \cap FV(T_1) = \varnothing$. Siano (σ, T) tali che $\sigma(\Gamma) \vdash \sigma(t_1 t_2) : T$. Per inversione abbiamo:

(i) $\sigma(\Gamma) \vdash \sigma(t_1) : T_1 \to T$;
(ii) $\sigma(\Gamma) \vdash \sigma(t_2) : T_1$;

Per ipotesi induttiva esistono:

(iii) $(\sigma_1, T_1 \to T)$ soluzione per $(\Gamma \,;\, t_1 \,;\, S_1 \,;\, \mathcal{C}_1)$;
(iv) (σ_2, T_1) soluzione per $(\Gamma \,;\, t_2 \,;\, S_2 \,;\, \mathcal{C}_2)$;

Inoltre σ_1 coincide con σ eccetto che su χ', mentre σ_2 coincide con σ eccetto che su χ''. Definiamo ora σ' come segue:

$$\sigma'(Y) \triangleq \begin{cases} T & \text{se } Y \equiv X \\ \sigma_1(Y) & \text{se } Y \in \chi' \\ \sigma_2(Y) & \text{se } Y \in \chi'' \\ \sigma(Y) & \text{altrimenti} \end{cases}$$

Osserviamo che questa è una buona definizione, perchè χ' e χ'' sono disgiunti, non contengono già X e non catturano le variabili di tipo di alcun termine. Ora σ':

(v) coincide con σ a meno che su χ;
(vi) soddisfa $\mathcal{C}_1 \cup \mathcal{C}_2$: infatti σ' soddisfa $\mathcal{C}_1$ perchè σ_1 soddisfa $\mathcal{C}_1$ e simmetricamente soddisfa $\mathcal{C}_2$ perchè quest'ultimo è soddisfatto da σ_2;
(vii) $\sigma'(X) \equiv T$;
(viii) $\sigma'(S_1) \equiv \sigma_1(S_1) \equiv T_1 \to T$;
(ix) $\sigma'(S_2) \equiv \sigma_2(S_2) \equiv T_1$.

Da (vii), (viii) e (ix) possiamo concludere che $\sigma'(S_1) \equiv \sigma'(S_2 \to X)$, quindi σ' soddisfa anche il vincolo $\{S_1 = S_2 \to X\}$.

– Altri casi: analoghi.

□

Definizione 2.32. *Date due sostituzioni σ e σ', σ è* più generale di *σ' ($\sigma \sqsubseteq \sigma'$) sse esiste una sostituzione ρ tale che $\sigma' = \sigma \circ \rho$.*

Definizione 2.33. *σ è* l'unificatore più generale (m.g.u.) *per $\mathcal{C}$ sse:*

i) σ *soddisfa* $\mathcal{C}$*;*

ii) per ogni σ' *che soddifa* $\mathcal{C}$*, si ha* $\sigma \sqsubseteq \sigma'$*.*

Figura 1 Algoritmo di unificazione di Robinson

```
unify(C)≜
    if C = ∅ then {}
        else let C' = C \ {S = T} in
            if S ≡ T then
                unify(C')
            else if S ≡ X ∧ X ∉ FV(T) then
                unify(C'[T/X]) ∘[X↦T]
            else if T ≡ X ∧ X ∉ FV(S) then
                unify(C'[S/X]) ∘[X↦S]
            else if S ≡ S1→S2 ∧ T ≡ T1→T2 then
                unify(C' ∪ {S1 = T1, S2 = T2})
            else fail.
```

Si dimostra che questo algoritmo termina sempre e sempre correttamente, restituendo `fail` se non esiste soluzione per $\mathcal{C}$, e l'mgu altrimenti.

Teorema 2.34. *Se* $(\Gamma\ ;\ t\ ;\ S\ ;\ \mathcal{C})$ *ha soluzione, allora unify(*$\mathcal{C}$*) restituisce* σ *tale che* $(\sigma, \sigma(S))$ *è la soluzione più generale, cioè per ogni soluzione* (σ', T') *si ha* $\sigma \sqsubseteq \sigma'$*.* $\sigma(S)$ *si dice* il tipo più generale di t dato Γ*.*

Come corollario abbiamo la decidibilità del problema di ricostruzione del tipo. Inoltre, se non disponiamo delle annotazioni di tipo sui λ è sempre possibile inserire delle variabili di tipo *fresche* sulle λ-astrazioni ed applicare lo stesso algoritmo visto in precedenza. La stessa cosa vale per i contesti: se abbiamo un termine t con $FV(t) = \{x_1, \ldots, x_n\}$, possiamo sempre costruire un contesto $\Gamma \triangleq \{x_1 : X_1, \ldots, x_n : X_n\}$ con X_i variabili di tipo *fresche*, e applicare l'algoritmo di ricostruzione.

Proposizione 2.35. *Dato un termine* $t \in \lambda\beta$ *è decidibile l'esistenza di un contesto* Γ*, un tipo* T *e un termine* t' *tali che* $\Gamma \vdash t' : T$ *e* $er(t') = t$*. Inoltre esiste un algoritmo che determina il* tipo principale *per* t*.*

Capitolo 3

Estensioni del calcolo

In questo capitolo estenderemo il linguaggio fin qui introdotto, muovendoci verso un vero linguaggio di programmazione. Partiamo da $\lambda\beta^T$ e introduciamo nuovi costrutti espandendo:

`[ M ]` : l'insieme dei *termini*;
`[ V ]` : l'insieme dei *valori*;
`[ E ]` : le *regole di valutazione*;
`[ S ]` : l'insieme dei *tipi*;
`[ T ]` : le *regole di tipizzazione*;

Quando alcuni di questi elementi verranno omessi, vorrà dire che non subiranno modifiche.

3.1 Ambienti locali

`[ M ]` $t ::= \ldots \mid \texttt{let}\ x = t\ \texttt{in}\ t$

`[ E ]` Siano v un valore e t un termine. La riduzione viene estesa con:

$$\texttt{let}\ x = v\ \texttt{in}\ t \to t[v/x] \quad (\textsc{Elet})$$

e con la regola di chiusura per contesti

$$\frac{t_1 \to t_1'}{\texttt{let}\ x = t_1\ \texttt{in}\ t_2 \to \texttt{let}\ x = t_1'\ \texttt{in}\ t_2}\ (\textsc{Elet}^{\text{cbv}})$$

`[ T ]` Alle regole di tipizzazione aggiungiamo:

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2 \qquad \Gamma \vdash t_1 : T_1}{\Gamma \vdash \texttt{let}\ x = t_1\ \texttt{in}\ t_2 : T_2}\ (\textsc{Tlet})$$

Il nuovo calcolo, esteso con questo costrutto, mantiene tutte le proprietà viste in precedenza. Questo discende dal fatto che "let–in" è *definibile* in $\lambda\beta^T$. Infatti:

$$\texttt{let}\ \ x = v\ \ \texttt{in}\ \ t \quad \triangleq \quad (\lambda x : S.t)v$$

dove S è il tipo del valore v. La computazione è concettualmente la stessa (la regola di valutazione di questa forma di "let–in" è del tutto analoga alla regola (Eβ) e la strategia di valutazione, grazie alla regola di chiusura per contesti, è weak call-by-value). La regola di tipo è derivabile come segue:

$$\frac{\Gamma, x : S \vdash t : T \qquad \Gamma \vdash v : S}{\Gamma \vdash \texttt{let}\ \ x = v\ \ \texttt{in}\ \ t : T} \quad \leftrightsquigarrow \quad \frac{\dfrac{\Gamma, x : S \vdash t : T}{\Gamma \vdash \lambda x : S.t : S{\to}T} \qquad \Gamma \vdash v : S}{\Gamma \vdash (\lambda x : S.t)v : T}$$

Osserviamo inoltre che questa forma di "let–in" è rigida rispetto ai tipi:

$$z : \texttt{Bool}{\to}\texttt{Nat}{\to}T \vdash \texttt{let}\ \ d = \lambda f{:}\_.\lambda x{:}\_.f(fx)\ \ \texttt{in}\ \ z(d\ (\lambda b{:}\texttt{Bool}.b)\ \texttt{tt})(d\ \texttt{succ}\ \underline{1})$$

non è tipizzabile. Quello che vorremmo fare è trasportare il polimorfismo "schematico" del metalivello dentro al linguaggio per poter tipizzare, ad esempio, questo termine. Cambiamo la regola di tipizzazione (T$_{\text{LET}}$):

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2[t_1/x] : T_2}{\Gamma \vdash \texttt{let}\ \ x = t_1\ \ \texttt{in}\ \ t_2 : T_2}\ (\text{T}_{\text{LET}}{}')$$

Occorre anche cambiare la regola di introduzione della freccia, eliminando l'annotazione di tipo nell'astrazione:

$$\frac{\Gamma, x : X \vdash t : T}{\Gamma \vdash \lambda x.t : X{\to}T}\ (\text{T}_{\text{ABS}}{}')$$

Quello che abbiamo ottenuto è noto in letteratura come "let polimorfo" o "let di Hindley-Milner" ed è la forma di ambiente locale del linguaggio ML (Milner et al. [1997]; Harper [2005]). Con questa forma di polimorfismo l'inferenza di tipo è, nel caso peggiore, esponenziale. Per ricostruire il tipo è obbligatorio esplicitare il contesto, per quanto questo sia un dettaglio poco significativo nella pratica, in cui i termini (programmi) che consideriamo sono sempre chiusi.

3.2 Il tipo singoletto

Il tipo singoletto o `Unit` è abitato da un solo elemento, il termine $*$, che è un valore.

`[ M ]` $t ::= \ldots \mid *$

[V] $v ::= \ldots \mid *$

[S] $T ::= \ldots \mid \texttt{Unit}$

[T] Alle regole di tipizzazione aggiungiamo l'assioma:

$$\Gamma \vdash * : \texttt{Unit} \quad (\mathrm{T}^*)$$

Il singoletto sta ad indicare un termine a cui non siamo interessati in quanto "valore di ritorno": infatti non dà nessuna informazione. Saremo bensì interessati ai side–effect delle computazioni che lo restituiscono.

3.3 Operatore di sequenza

È il classico operatore che permette di concatenare più istruzioni, espresso in molti linguaggi di programmazione mediante il punto e virgola.

[M] $t ::= \ldots \mid t; t$

[E] Aggiungiamo una regola di semplificazione:

$$*; t_2 \to t_2 \quad (\textsc{Eseq})$$

ed un'altra per esplicitare l'ordine di valutazione:

$$\frac{t_1 \to t_1'}{t_1; t_2 \to t_1'; t_2} \, (\textsc{Eseq}^{\mathrm{LX}})$$

[T] Alle regole di tipizzazione aggiungiamo:

$$\frac{\Gamma \vdash t_1 : \texttt{Unit} \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1; t_2 : T_2} \, (\textsc{Tseq})$$

In pratica il valore ritornato da una sequenza di termini è lo stesso dell'ultimo termine nella sequenza, a patto che tutti quelli prima di lui restituiscano sempre $*$.

In alternativa a questa formulazione, si può pensare di non effettuare una vera e propria estensione del calcolo, bensì di aggiungere un *alias* così definito:

$$t_1; t_2 \quad \triangleq \quad (\lambda x : \texttt{Unit}.t_2)t_1 \;\; \text{con } x \notin FV(t_2)$$

Ora la regola di tipizzazione (Tseq) è derivabile:

$$\frac{\Gamma \vdash t_1 : \texttt{Unit} \qquad \dfrac{\dfrac{\Gamma \vdash t_2 : T_2}{\Gamma, x : \texttt{Unit} \vdash t_2 : T_2} \text{ poichè } x \notin FV(t_2)}{\Gamma \vdash \lambda x : \texttt{Unit}.t_2 : \texttt{Unit} \to T_2}}{\Gamma \vdash (\lambda x : \texttt{Unit}.t_2)t_1 : T_2}$$

3.4 Annotazioni esplicite di tipo

Talvolta può essere utile esplicitare il tipo di un'espressione (le motivazioni saranno più evidenti quando parleremo di *somme* e di *sottotipi e cast*). Le annotazioni di tipo (o *ascription*) svolgono precisamente questa funzione.

[M] $t ::= \ldots \mid t \texttt{ as } T$

[E] Sia v un valore:

$$v \texttt{ as } T \to v \quad (\text{E}_{\text{AS}})$$

Regola contestuale:

$$\frac{t \to t'}{t \texttt{ as } T \to t' \texttt{ as } T} \; (\text{E}_{\text{AS}}{}^{\text{ctx}})$$

[T] Alle regole di tipizzazione aggiungiamo:

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash t \texttt{ as } T : T} \; (\text{T}_{\text{AS}})$$

3.5 Prodotti e somme

I prodotti corrispondono alle *coppie* o, nell'isomorfismo di Curry–Howard, alle *congiunzioni logiche*, le somme alle *unioni (disgiunte)* o, via Curry–Howard, alle *disgiunzioni*.

[M] Ai termini aggiungiamo un nuovo costruttore di coppie e due distruttori, rispettivamente:

$$t ::= \ldots \mid \texttt{(}t\texttt{,}t\texttt{)} \mid \texttt{fst } t \mid \texttt{snd } t$$

mentre, in maniera del tutto simmetrica, per le somme abbiamo due nuovi costruttori ed un distruttore:

$$t ::= \ldots \mid \texttt{inl } t \mid \texttt{inr } t \mid \texttt{case } t \texttt{ of inl } x \Rightarrow t \texttt{ | inr } x \Rightarrow t$$

[V] Le coppie di valori sono a loro volta valori, così come lo sono le iniezioni destre e sinistre di valori:

$$v ::= \ldots \mid \texttt{(}v\texttt{,}v\texttt{)} \mid \texttt{inl } v \mid \texttt{inr } v$$

[E] Siano v_1, v_2 valori. Per le coppie, abbiamo le seguenti regole di valutazione che, in coerenza con la strategia *call-by-value*, impongono la riduzione a valori degli argomenti degli operatori `fst` e `snd`, e che forzano la valutazione delle componenti della coppia da sinistra a destra:

$$\texttt{fst}\ (v_1\texttt{,}v_2) \ \to\ v_1 \quad (\textsc{Efst})$$
$$\texttt{snd}\ (v_1\texttt{,}v_2) \ \to\ v_2 \quad (\textsc{Esnd})$$

$$\frac{t \to t'}{\texttt{fst}\ t \to \texttt{fst}\ t'}\ (\textsc{Efst}^{\text{ctx}}) \qquad \frac{t \to t'}{\texttt{snd}\ t \to \texttt{snd}\ t'}\ (\textsc{Esnd}^{\text{ctx}})$$

$$\frac{t_1 \to t_1'}{(t_1\texttt{,}t_2) \to (t_1'\texttt{,}t_2)}\ (\textsc{Epair}^{\text{LX}}) \qquad \frac{t_2 \to t_2'}{(v_1\texttt{,}t_2) \to (v_1\texttt{,}t_2')}\ (\textsc{Epair}^{\text{RX}})$$

Per le somme abbiamo due assiomi per il distruttore, corredati dalle regole contestuali coerenti con la strategia per valore:

$$\texttt{case}\ (\texttt{inl}\ v_1)\ \texttt{of inl}\ x \Rightarrow t_1\ \texttt{|}\ \texttt{inr}\ x \Rightarrow t_2 \ \to\ t_1[v_1/x] \quad (\textsc{Ecase}^{\text{inl}})$$
$$\texttt{case}\ (\texttt{inr}\ v_1)\ \texttt{of inl}\ x \Rightarrow t_1\ \texttt{|}\ \texttt{inr}\ x \Rightarrow t_2 \ \to\ t_2[v_1/x] \quad (\textsc{Ecase}^{\text{inr}})$$

$$\frac{t \to t'}{\texttt{inl}\ t \to \texttt{inl}\ t'}\ (\textsc{Einl}^{\text{ctx}}) \qquad \frac{t \to t'}{\texttt{inr}\ t \to \texttt{inr}\ t'}\ (\textsc{Einr}^{\text{ctx}})$$

$$\frac{t \to t'}{\texttt{case}\ t\ \texttt{of}\ \ldots \to \texttt{case}\ t'\ \texttt{of}\ \ldots}\ (\textsc{Ecase}^{\text{ctx}})$$

[S] Al solito, i tipi ci servono per non ritrovarci bloccati su forme normali che non sono valori. Come detto in precedenza, prodotti e somme sono l'analogo via Curry–Howard di congiunzioni e disgiunzioni logiche. Il sistema di tipi viene dunque esteso con due nuovi costruttori di tipo:

$$T ::= \ldots \mid T{\times}T \mid T{+}T$$

[T] Per i prodotti aggiungiamo l'analogo delle regole di introduzione e di eliminazione (sinistra e destra) del connettivo logico di congiunzione:

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash (t_1\texttt{,}t_2) : T_1{\times}T_2}\ (\textsc{T}\times)$$

$$\frac{\Gamma \vdash t : T_1{\times}T_2}{\Gamma \vdash \texttt{fst}\ t : T_1}\ (\textsc{Tfst}) \qquad \frac{\Gamma \vdash t : T_1{\times}T_2}{\Gamma \vdash \texttt{snd}\ t : T_2}\ (\textsc{Tsnd})$$

Mentre per le somme avremo:

$$\frac{\Gamma \vdash t : T_1}{\Gamma \vdash \texttt{inl}\ t : T_1{+}T_2}\ (\textsc{Tinl}) \qquad \frac{\Gamma \vdash t : T_2}{\Gamma \vdash \texttt{inr}\ t : T_1{+}T_2}\ (\textsc{Tinr})$$

$$\frac{\Gamma \vdash t : T_1{+}T_2 \qquad \Gamma, x : T_1 \vdash t_1 : T \qquad \Gamma, x : T_2 \vdash t_2 : T}{\Gamma \vdash \texttt{case}\ t\ \texttt{of inl}\ x \Rightarrow t_1\ \texttt{|}\ \texttt{inr}\ x \Rightarrow t_2 : T}\ (\textsc{Tcase})$$

Con le modifiche fatte finora, il calcolo gode ancora delle proprietà di normalizzazione e di subject–reduction. Abbiamo però perso l'unicità del tipo, a causa delle regole (Tinl) e (Tinr): infatti, nel primo caso T_2 può essere qualunque tipo, e la stessa cosa vale per T_1 nel caso di (Tinr). Per ripristinare questa proprietà si può modificare la sintassi di `inl` e `inr`, rendendo esplicito il tipo in cui i termini vengono iniettati:

$$t ::= \ldots \mid \texttt{inl}\ t\ \texttt{as}\ T \mid \texttt{inr}\ t\ \texttt{as}\ T$$

Ora, modificando le regole di tipizzazione (Tinl) e (Tinr) come segue, riotteniamo l'unicità del tipo:

$$\frac{\Gamma \vdash t : T_1}{\Gamma \vdash \texttt{inl}\ t\ \texttt{as}\ T_1{+}T_2 : T_1{+}T_2}\ (\textsc{Tinl}) \qquad \frac{\Gamma \vdash t : T_2}{\Gamma \vdash \texttt{inr}\ t\ \texttt{as}\ T_1{+}T_2 : T_1{+}T_2}\ (\textsc{Tinr})$$

3.6 Record

Nella pratica dei linguaggi di programmazione, prodotti e somme assumono forme diverse, *etichettate*, i *record* ed i *varianti*; le etichette costituiscono una notevole *facility* per i programmatori che utilizzano questi costrutti.

I record sono prodotti n–ari (o *tuple*), in cui possiamo accedere ai campi grazie ad un *nome* (l'etichetta) anzichè proiettando la componente d'interesse mediante un indice intero.

Sia $\mathcal{L}$ un insieme infinito numerabile di etichette.

[M] Siano $\{l, l_1, \ldots, l_n\} \subseteq \mathcal{L}$. La sintassi viene estesa con un costruttore ed un distruttore di record:

$$t ::= \ldots \mid \{l_1{=}t, \ldots, l_n{=}t\} \mid t.l$$

[V] $v ::= \ldots \mid \{l_1{=}v, \ldots, l_n{=}v\}$

[E] Siano $v_1, \ldots, v_n$ valori, $t, t_1, \ldots, t_n$ termini e $i, j \in \{1, \ldots, n\}$:

$$\{l_1{=}v_1\,,\ldots,l_n{=}v_2\}\,.\,l_i \to v_i\ (\textsc{Esel}) \qquad \frac{t \to t'}{t\,.\,l \to t'\,.\,l}\ (\textsc{Esel}^{\text{ctx}})$$

$$\frac{t_j \to t'_j}{\{l_1{=}v_1\,,\ldots,l_{j-1}{=}v_{j-1}\,,l_j{=}t_j\,,\ldots,l_n{=}t_n\} \to \{l_1{=}v_1\,,\ldots,l_{j-1}{=}v_{j-1}\,,l_j{=}t'_j\,,\ldots,l_n{=}t_n\}}\ (\textsc{Erec})$$

[S] $T ::= \ldots \mid \{l_1{:}T\,,\ldots,l_n{:}T\}$

[T] Alle regole di tipizzazione aggiungiamo:

$$\frac{\Gamma \vdash t_1 : T_1 \quad \ldots \quad \Gamma \vdash t_n : T_n}{\Gamma \vdash \{l_1{=}t_1\,,\ldots,l_n{=}t_n\} : \{l_1{:}T_1\,,\ldots,l_n{:}T_n\}}\ (\textsc{Trec}) \qquad \frac{\Gamma \vdash t : \{l_1{:}T_1\,,\ldots,l_n{:}T_n\}}{\Gamma \vdash t\,.\,l_i : T_i}\ (\textsc{Tsel})$$

3.7 Varianti

I *varianti* sono *somme etichettate*, cioè la generalizzazione delle unioni disgiunte, con l'aggiunta delle etichette. Siano $\{l, l_1, \ldots, l_n\} \subseteq \mathcal{L}$.

[M]

$$\begin{array}{rcll} t ::= \ldots & | & \texttt{<}l\texttt{=}t\texttt{>}\ \texttt{as}\ T & \\ & | & \texttt{case}\ t\ \ \texttt{of} & \texttt{<}l_1\texttt{=}x\texttt{>} \Rightarrow t_1 \\ & & \quad\ \texttt{|} & \texttt{<}l_2\texttt{=}x\texttt{>} \Rightarrow t_2 \\ & & \quad \ldots & \quad \ldots \\ & & \quad\ \texttt{|} & \texttt{<}l_n\texttt{=}x\texttt{>} \Rightarrow t_n \end{array}$$

[V] $v ::= \ldots \mid \texttt{<}l\texttt{=}v\texttt{>}\ \texttt{as}\ T$

[E] Siano $v_1, \ldots, v_n$ valori, $t, t', t_1, \ldots, t_n$ termini e $i \in \{1, \ldots, n\}$:

$$\begin{array}{lll} \texttt{case}\ \texttt{<}l_i\texttt{=}v_i\texttt{>}\ \texttt{as}\ T\ \ \texttt{of} & \texttt{<}l_1\texttt{=}x\texttt{>} \Rightarrow t_1 & \\ \quad\ \texttt{|} & \texttt{<}l_2\texttt{=}x\texttt{>} \Rightarrow t_2 & \\ \quad \ldots & \quad \ldots & \\ \quad\ \texttt{|} & \texttt{<}l_n\texttt{=}x\texttt{>} \Rightarrow t_n \ \ \to \ \ t_i[v_i/x] & (\textsc{Ecase}^{\text{n}}) \end{array}$$

$$\frac{t \to t'}{\texttt{case}\ t\ \texttt{as}\ T\ \texttt{of}\ \ \ldots \to \texttt{case}\ t'\ \texttt{as}\ T\ \texttt{of}\ \ \ldots}\ (\textsc{Ecase}^{\text{nctx}})$$

$$\frac{t \to t'}{\texttt{<}l\texttt{=}t\texttt{>}\ \texttt{as}\ T \to \texttt{<}l\texttt{=}t'\texttt{>}\ \texttt{as}\ T}\ (\textsc{Evrn}^{\text{ctx}})$$

[S] $T ::= \ldots \mid \texttt{<}l_1{:}T\,,\ldots,l_n{:}T\texttt{>}$

[T] Sia $i \in \{1, \ldots, n\}$. Alle regole di tipizzazione aggiungiamo:

$$\frac{\Gamma \vdash t : T_i}{\Gamma \vdash \texttt{<}l_i\texttt{=}t\texttt{>}\ \texttt{as}\ \texttt{<}l_1\texttt{:}T_1\texttt{,}\ldots\texttt{,}l_n\texttt{:}T_n\texttt{>} : \texttt{<}l_1\texttt{:}T_1\texttt{,}\ldots\texttt{,}l_n\texttt{:}T_n\texttt{>}} \ (\textsc{T}_{\textsc{vrn}})$$

$$\frac{\Gamma \vdash t : \texttt{<}l_1\texttt{:}T_1\texttt{,}\ldots\texttt{,}l_n\texttt{:}T_n\texttt{>} \qquad \Gamma, x : T_1 \vdash t_1 : S \quad \ldots \quad \Gamma, x : T_n \vdash t_n : S}{\begin{array}{rcl} \Gamma \vdash \texttt{case}\ t & \texttt{of} & \texttt{<}l_1\texttt{=}x\texttt{>} \Rightarrow t_1 \\ & \texttt{|} & \texttt{<}l_2\texttt{=}x\texttt{>} \Rightarrow t_2 \\ & \ldots & \ldots \\ & \texttt{|} & \texttt{<}l_n\texttt{=}x\texttt{>} \Rightarrow t_n \ : \ S \end{array}} \ (\textsc{T}_{\textsc{sel}})$$

Seguono alcuni esempi di utilizzo dei varianti:

- Opzioni. Vogliamo definire una funzione f che prende in input un intero e ritorna un naturale oppure "errore". Grazie ai varianti possiamo scrivere:
$$f : \texttt{Nat} \to \texttt{<n:Nat, e:Unit>}$$
- Enumerazioni. Il tipo enumerativo:
$$\text{Settimana} = \text{Lun}, \text{Mar}, \ldots, \text{Dom}$$
sarà codificato con:
$$\text{Sett} \triangleq \texttt{<Lun:Unit, \ldots, Dom:Unit>}$$
In questo variante non siamo interessati ai valori contenuti (che infatti sono tutti $*$, unico abitante del tipo `Unit`), bensì alle etichette. Ad esempio, la funzione che calcola il prossimo giorno della settimana può essere così codificata:
$$\begin{array}{rcl} \text{next} \triangleq \lambda s : \underline{\text{Sett}}.\texttt{case}\ s & \texttt{of} & \texttt{<Lun=}x\texttt{>} \Rightarrow \texttt{<Mar=*>}\ \texttt{as}\ \underline{\text{Sett}} \\ & \texttt{|} & \texttt{<Mar=}x\texttt{>} \Rightarrow \texttt{<Mer=*>}\ \texttt{as}\ \underline{\text{Sett}} \\ & \ldots & \ldots \\ & \texttt{|} & \texttt{<Dom=}x\texttt{>} \Rightarrow \texttt{<Lun=*>}\ \texttt{as}\ \underline{\text{Sett}} \end{array}$$

3.8 Liste

Utilizziamo una forma di polimorfismo schematico per estendere il linguaggio con le "liste di tipo T".

[M] $t ::= \ldots \mid \texttt{nil}_T \mid \texttt{cons}_T\ t\ t \mid \texttt{hd}_T\ t \mid \texttt{tl}_T\ t \mid \texttt{isnil}_T\ t$

[V] $v ::= \ldots \mid \texttt{nil}_T \mid \texttt{cons}_T\ v\ v$

[E] Siano v_1, v_2 valori:

$$\texttt{isnil}_S\ \texttt{nil}_T \to \texttt{tt} \quad (\textsc{Eisnil}^{\text{tt}})$$

$$\texttt{isnil}_S\ (\texttt{cons}_T\ v_1\ v_2) \to \texttt{ff} \quad (\textsc{Eisnil}^{\text{ff}})$$

$$\frac{t \to t'}{\texttt{isnil}_T\ t \to \texttt{isnil}_T\ t'}\ (\textsc{Eisnil}^{\text{ctx}})$$

$$\frac{t_1 \to t_1'}{\texttt{cons}_T\ t_1\ t_2 \to \texttt{cons}_T\ t_1'\ t_2}\ (\textsc{Econs}^{\text{LX}}) \qquad \frac{t_2 \to t_2'}{\texttt{cons}_T\ v_1\ t_2 \to \texttt{cons}_T\ v_1\ t_2'}\ (\textsc{Econs}^{\text{RX}})$$

$$\texttt{head}_S\ (\texttt{cons}_T\ v_1\ v_2) \to v_1 \quad (\textsc{Ehead})$$

$$\texttt{tail}_S\ (\texttt{cons}_T\ v_1\ v_2) \to v_2 \quad (\textsc{Etail})$$

$$\frac{t \to t'}{\texttt{head}_T\ t \to \texttt{head}_T\ t'}\ (\textsc{Ehead}^{\text{ctx}}) \qquad \frac{t \to t'}{\texttt{tail}_T\ t \to \texttt{tail}_T\ t'}\ (\textsc{Etail}^{\text{ctx}})$$

Infine, per preservare il *teorema di progresso* dobbiamo aggiungere le regole:

$$\texttt{head}_S\ \texttt{nil}_T \to \texttt{nil}_T \quad (\textsc{Ehead}^{\text{nil}})$$

$$\texttt{tail}_S\ \texttt{nil}_T \to \texttt{nil}_T \quad (\textsc{Etail}^{\text{nil}})$$

che sono analoghe alla regola $(\textsc{Epred}^0)$ nel calcolo delle espressioni aritmetiche e booleane.

[S] $T ::= \ldots \mid \texttt{list}\ T$

[T]

$$\Gamma \vdash \texttt{nil}_T : \texttt{list}\ T \quad (\textsc{Tnil})$$

$$\frac{\Gamma \vdash t_1 : T \qquad \Gamma \vdash t_2 : \texttt{list}\ T}{\Gamma \vdash \texttt{cons}_T\ t_1\ t_2 : \texttt{list}\ T}\ (\textsc{Tcons}) \qquad \frac{\Gamma \vdash t : \texttt{list}\ T}{\Gamma \vdash \texttt{isnil}_T\ t : \texttt{Bool}}\ (\textsc{Tisnil})$$

$$\frac{\Gamma \vdash t : \texttt{list}\ T}{\Gamma \vdash \texttt{hd}_T\ t : T}\ (\textsc{Thd}) \qquad \frac{\Gamma \vdash t : \texttt{list}\ T}{\Gamma \vdash \texttt{tl}_T\ t : \texttt{list}\ T}\ (\textsc{Ttl})$$

3.9 Ricorsione generale

In lambda calcolo puro ($\lambda\beta$ con full-beta-reduction) il termine:

$$Y \triangleq \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

esprime la ricorsione come punto fisso di un funzionale.

Vogliamo aggiungere un operatore che "simuli" nel calcolo il comportamento del termine Y.

[M] $t ::= \ldots \mid \texttt{fix}\ t$

[E]

$$\texttt{fix}\ (\lambda x : T.t) \to t[\texttt{fix}\ (\lambda x : T.t)/x] \quad (\textsc{E}_{\textsc{FIX}})$$

$$\frac{t \to t'}{\texttt{fix}\ t \to \texttt{fix}\ t'}\ \textsc{E}_{\textsc{FIX}}{}^{\text{ctx}}$$

Osserviamo che in queste regole si trova una forte giustificazione del fatto che abbiamo scelto fin dall'inizio di considerare le lambda-astrazioni come valori, e quindi di non ridurre mai sotto un λ.

[T]

$$\frac{\Gamma \vdash t : T \to T}{\Gamma \vdash \texttt{fix}\ t : T}\ (\textsc{T}_{\textsc{FIX}})$$

Osserviamo che il termine:

$$\text{Su} \triangleq \lambda n : \texttt{Nat.succ}\ n\ :\ \texttt{Nat} \to \texttt{Nat}$$

applicato all'operatore `fix` produce divergenza:

$$\texttt{fix}\ \underline{\text{Su}} \to \texttt{succ}\ (\texttt{fix}\ \underline{\text{Su}}) \to \cdots$$

e più in generale, per ogni tipo T, il termine:

$$\text{div}_T \triangleq \lambda x : \texttt{Unit.fix}\ (\lambda x : T.x)\ :\ \texttt{Unit} \to T$$

applicato a $*$ diverge. Inoltre $(\underline{\text{div}_T}\ *)$ è di tipo T per ogni T, quindi nel calcolo che abbiamo ottenuto ogni tipo è abitato da almeno un termine e questo rompe la corrispondenza di Curry–Howard (non è vero che ogni formula logica proposizionale è dimostrabile).

Grazie a questa forma di ricorsione è possibile esprimere il comune costrutto *letrec*:

$$\texttt{letrec}\ f = \lambda x : \_.t_1\ \texttt{in}\ t_2$$

dove f può comparire all'interno di t_1. *letrec* viene tradotto in:

$$\texttt{let } f = \texttt{fix } (\lambda f : \_.\lambda x : \_.t_1) \texttt{ in } t_2$$

ottenendone una formulazione equivalente che utilizza solamente l'operatore `fix` di punto fisso. Come nel caso del "let-in", è interessante osservare che ci sono due modi di vedere questa forma di definizione ricorsiva: si può pensare che in t_1, la funzione f possa essere usata *solo* in modo monomorfo (questa è la politica di ML); oppure si può pensare ad una forma più generale di "letrec polimorfo", che tuttavia induce un problema di ricostruzione indecidibile.

Consideriamo le funzioni (scritte in pseudocodice):

$$\begin{aligned} \text{pari?}(n) &\triangleq \text{if}(n = 0) \text{ then tt else } (\text{disp?}(\text{pred } n)) \\ \text{disp?}(n) &\triangleq \text{if}(n = 0) \text{ then ff else } (\text{pari?}(\text{pred } n)) \end{aligned}$$

mutuamente ricorsive. Definiamo il funzionale F come segue:

$$\begin{aligned} \text{F} \triangleq\ & \lambda re : \{apari : \texttt{Nat}\to\texttt{Bool},\ adisp : \texttt{Nat}\to\texttt{Bool}\}. \\ & \{apari{=}\lambda n : \texttt{Nat.if } (\texttt{iszero } n) \texttt{ then tt else } (re.adisp\ (\texttt{pred } n)) \\ & \ adisp{=}\lambda n : \texttt{Nat.if } (\texttt{iszero } n) \texttt{ then ff else } (re.apari\ (\texttt{pred } n))\} \end{aligned}$$

e osserviamo che:

$$\texttt{fix } \underline{\text{F}} : \{apari : \texttt{Nat}\to\texttt{Bool},\ adisp : \texttt{Nat}\to\texttt{Bool}\}$$

È possibile dimostrare che:

$$\begin{aligned} \text{pari?} &\simeq (\texttt{fix } F).apari \\ \text{disp?} &\simeq (\texttt{fix } F).adisp \end{aligned}$$

3.10 Locazioni

Aggiungiamo al linguaggio le variabili accessibili in lettura e scrittura, tipiche dei linguaggi imperativi.

Prendiamo un insieme infinito numerabile di *locazioni* $\mathcal{H}$. Le locazioni denotano il "posto" in cui i valori sono contenuti, sono l'analogo dei *nomi* delle variabili nei linguaggi imperativi. Sarà necessario definire un concetto di *memoria* o *stato*, cioè una funzione:

$$\mu : \mathcal{H}\to\text{Valori}$$

che associa locazioni a valori.

[M] Dobbiamo estendere la sintassi del linguaggio con il *costruttore* per le variabili, gli operatori di *assegnamento* e *accesso al contenuto* delle variabili, e le locazioni.

$$t ::= \ldots \mid \texttt{ref}\ \ t \mid t\texttt{:=}t \mid \texttt{!}t \mid l$$

[V] $v ::= \ldots \mid l$

[E] Nella valutazione il formalismo deve essere esteso con le memorie. Ad esempio la regola (Eβ) diventa:

$$(\lambda x : T.t)v \mid \mu \quad \rightarrow \quad t[v/x] \mid \mu$$

cioè la memoria iniziale non viene modificata dal passo di valutazione. Aggiungiamo le seguenti regole di valutazione:

$$\texttt{ref}\ \ v \mid \mu \rightarrow l \mid \mu[l \mapsto v] \quad \text{con l “fresca”} \quad (\text{E}_{\text{REF}})$$

$$\texttt{!}v \mid \mu \rightarrow \mu(v) \mid \mu \quad (\text{E}_{\text{BANG}})$$

Osserviamo che in quest'ultima regola, il fatto che v sia una locazione verrà garantito dalle regole di tipizzazione. Un discorso analogo vale anche per v_1 in:

$$v_1\texttt{:=}v_2 \mid \mu \rightarrow * \mid \mu[v_1 \mapsto v_2] \quad (\text{E}_{\text{ASN}})$$

$$\frac{t \mid \mu \rightarrow t' \mid \mu'}{\texttt{ref}\ \ t \mid \mu \rightarrow \texttt{ref}\ \ t' \mid \mu'}\ (\text{E}_{\text{REF}}{}^{\text{ctx}}) \qquad \frac{t \mid \mu \rightarrow t' \mid \mu'}{\texttt{!}t \mid \mu \rightarrow \texttt{!}t' \mid \mu'}\ (\text{E}_{\text{BANG}}{}^{\text{ctx}})$$

$$\frac{t_1 \mid \mu \rightarrow t_1' \mid \mu'}{t_1\texttt{:=}t_2 \mid \mu \rightarrow t_1'\texttt{:=}t_2 \mid \mu'}\ (\text{E}_{\text{ASN}}{}^{\text{LX}}) \qquad \frac{t_2 \mid \mu \rightarrow t_2' \mid \mu'}{v\texttt{:=}t_2 \mid \mu \rightarrow v\texttt{:=}t_2' \mid \mu'}\ (\text{E}_{\text{ASN}}{}^{\text{RX}})$$

[S] $T ::= \ldots \mid \texttt{Ref}\ \ T$

[T] Per tipizzare le locazioni, abbiamo bisogno di una funzione ausiliaria:

$$\Sigma : \mathcal{H} \rightarrow \text{Tipi}$$

che associa ogni locazione al tipo del valore ivi contenuto. I giudizi di tipo prenderanno la forma $\Sigma, \Gamma \vdash t : T$. Aggiungiamo:

$$\frac{\Sigma, \Gamma \vdash t : T}{\Sigma, \Gamma \vdash \texttt{ref}\ \ t : \texttt{Ref}\ \ T}\ (\text{T}_{\text{REF}}) \qquad \frac{\Sigma, \Gamma \vdash t : \texttt{Ref}\ \ T}{\Sigma, \Gamma \vdash \texttt{!}t : T}\ (\text{T}_{\text{BANG}})$$

$$\frac{\Sigma, \Gamma \vdash t_1 : \texttt{Ref}\ T \qquad \Sigma, \Gamma \vdash t_2 : T}{\Sigma, \Gamma \vdash t_1 \texttt{:=} t_2 : \texttt{Unit}}\ (\textsc{T}_{\textsc{asn}}) \qquad \frac{l : T \in \Sigma}{\Sigma, \Gamma \vdash l : \texttt{Ref}\ T}\ (\textsc{T}_{\textsc{loc}})$$

Il linguaggio che abbiamo ottenuto, pur senza l'operatore di punto fisso, rompe il teorema di forte normalizzazione. Ad esempio il termine:

$$\begin{array}{l}\texttt{let a = ref } (\lambda \text{x:}\texttt{Nat}.\underline{0})\ \texttt{in} \\ \quad \texttt{let b = ref } (\lambda \text{x:}\texttt{Nat}.(\texttt{!a})\,\text{x})\ \texttt{in} \\ \quad\quad \text{a}\texttt{:=}(\lambda \text{x:}\texttt{Nat}.(\texttt{!b})\,\text{x})\texttt{;}\ (\texttt{!b})\,\underline{0}\end{array}$$

è divergente. È possibile dimostrare che la ricorsione generale è ottenibile solo con l'uso delle variabili.

Definizione 3.1. *La memoria* μ realizza Σ, Γ *(o anche* μ è corretta rispetto Σ, Γ*, in simboli:* $\Sigma, \Gamma \vdash \mu$*) sse:*

(1) $\mathrm{dom}(\mu) = \mathrm{dom}(\Sigma)$

(2) $\forall l \in \mathrm{dom}(\mu).\Sigma, \Gamma \vdash \mu(l) : \Sigma(l)$

Teorema 3.2 (Subject reduction)**.** *Se:*

$$\Sigma, \Gamma \vdash t : T \quad e \quad \Sigma, \Gamma \vdash \mu \quad e \quad t \mid \mu \to t' \mid \mu'$$

allora esiste $\Sigma' \supseteq \Sigma$ *tale che:*

$$\Sigma', \Gamma \vdash t' : T \quad e \quad \Sigma', \Gamma \vdash \mu'$$

Dimostrazione. Per induzione sulla derivazione di $t \mid \mu \to t' \mid \mu'$ usando il lemma di inversione ed i seguenti lemmi ausiliari:

- *Lemma (Sostituzione).* Sia x una variabile:

  $$\text{se} \quad \Sigma, \Gamma, x : S \vdash t : T \quad \text{e} \quad \Sigma, \Gamma \vdash s : S \quad \text{allora} \quad \Sigma, \Gamma \vdash t[s/x] : T$$

- *Lemma (Sostituzione su memorie).* Sia l una locazione:

  $$\text{se} \quad \Sigma, \Gamma \vdash \mu \quad \text{e} \quad \Sigma(l) \equiv T \quad \text{e} \quad \Sigma, \Gamma \vdash v : T \quad \text{allora} \quad \Sigma, \Gamma \vdash \mu[l \mapsto v]$$

- *Lemma.*

  $$\text{se} \quad \Sigma, \Gamma \vdash t : T \quad \text{e} \quad \Sigma' \supseteq \Sigma \quad \text{allora} \quad \Sigma', \Gamma \vdash t : T$$

□

Teorema 3.3 (Progresso)**.** *Sia t chiuso e ben tipizzato, cioè:*

$$\Sigma, \emptyset \vdash t : T \quad \textit{per qualche } \Sigma \textit{ e } T$$

Allora t è un valore oppure per ogni μ tale che $\Sigma, \emptyset \vdash \mu$ esistono t', μ' tali che:

$$t \mid \mu \to t' \mid \mu'$$

Dimostrazione. Per induzione sulla derivazione di tipo $\Sigma, \emptyset \vdash t : T$, usando il lemma di forme canoniche. □

Il calcolo è *safe* ma la normalizzazione non è preservata, come già osservato. Infatti si può mostrare che una definizione ricorsiva di funzione della forma:

$$f \triangleq \mathcal{E}[f]$$

può essere codificata ("risolta") in un termine con locazioni.

3.11 Errori ed eccezioni

Estendiamo il linguaggio in modo da tenere in conto situazioni di errore, e di poterle gestire (catturare) mediante un nuovo costrutto "try–with".

[M] $t ::= \ldots \mid \texttt{error} \mid \texttt{try}\ t\ \texttt{with}\ t$

[V] Non ci sono nuovi valori da introdurre. Infatti `error` è da considerarsi come una funzione, non come un valore.

[E] Sia v un valore:

$$\texttt{error}\ t \to \texttt{error} \quad (\textsc{Err}^{\textsc{Lx}})$$

$$v\ \texttt{error} \to \texttt{error} \quad (\textsc{Err}^{\textsc{Rx}})$$

$$\texttt{try error with}\ t \to t \quad (\textsc{With})$$

$$\texttt{try}\ v\ \texttt{with}\ t \to v \quad (\textsc{Try})$$

$$\frac{t_1 \to t_1'}{\texttt{try}\ t_1\ \texttt{with}\ t_2 \to \texttt{try}\ t_1'\ \texttt{with}\ t_2} \ (\textsc{Try}^{\textsc{ctx}})$$

[S] $\Gamma \vdash \texttt{error} : T$, cioè `error` abita ogni tipo.

[T]

$$\frac{\Gamma \vdash t_1 : T \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash \texttt{try}\ t_1\ \texttt{with}\ t_2 : T} \ (\textsc{T}_{\textsc{Try}})$$

Infine bisogna modificare il teorema di progresso per tener conto degli errori; i casi, dato t chiuso e ben tipizzato, diventano:

- t è un valore, oppure;
- t è `error`, oppure;
- esiste t' tale che $t \to t'$;

Capitolo 4

Sottotipi

La *relazione di sottotipo* o di *compatibilità*:

$$S <: T$$

ci dice che ogni termine di tipo S può essere utilizzato come termine di tipo T, o che S *è sottotipo* di T, o anche che T *è sovratipo* di S.

Vogliamo definire delle relazioni tra tipi primitivi (cioè assiomi per $<:$) e studiare come i costruttori di tipo (e.g. $\to$) si comportano rispetto alla relazione di sottotipo. Le proprietà generali della relazione sono:

- *Riflessività*: ogni tipo è compatible con se stesso.

$$T <: T \quad (\textsc{Srefl})$$

- *Transitività*.

$$\frac{R <: S \qquad S <: T}{R <: T} \ (\textsc{Stran})$$

- *Sussunzione* o *subsumption*: questa regola rappresenta il punto di contatto tra la relazione di tipizzazione vista fin qui e la relazione di sottotipo.

$$\frac{\Gamma \vdash t : S \qquad S <: T}{\Gamma \vdash t : T} \ (\textsc{Tsub})$$

Mentre per i record abbiamo le seguenti regole:

- *Compatiblità in ampiezza*.

$$\{l_1 : T_1 , \ldots , l_n : T_n , l_{n+1} : T_{n+1} , \ldots , l_{n+m} : T_{n+m}\} <: \{l_1 : T_1 , \ldots , l_n : T_n\} \quad (\textsc{Srec}^{\text{a}})$$

- *Compatiblità in profondità*.

$$\frac{T_1 <: S_1 \qquad \cdots \qquad T_n <: S_n}{\{l_1 : T_1 , \ldots , l_n : T_n\} <: \{l_1 : S_1 , \ldots , l_n : S_n\}} \ (\textsc{Srec}^{\text{p}})$$

- *Permutazione.* Sia π una permutazione da $\{1, ..., n\}$ a $\{1, ..., n\}$:

$$\{l_1 : T_1 , ... , l_n : T_n\} <: \{l_{\pi(1)} : T_{\pi(1)} , ... , l_{\pi(n)} : T_{\pi(n)}\} \quad (\textsc{Srec}^{\pi})$$

Estendiamo il linguaggio introducendo un nuovo tipo predefinito, il tipo `Top`. Gli oggetti (i termini) di tipo `Top` sono i meno definiti e non hanno operatori applicabili. Per ogni tipo T è vero che:

$$T <: \texttt{Top} \quad (\textsc{Stop})$$

cioè `Top` è *l'elemento massimo* della relazione si sottotipo.

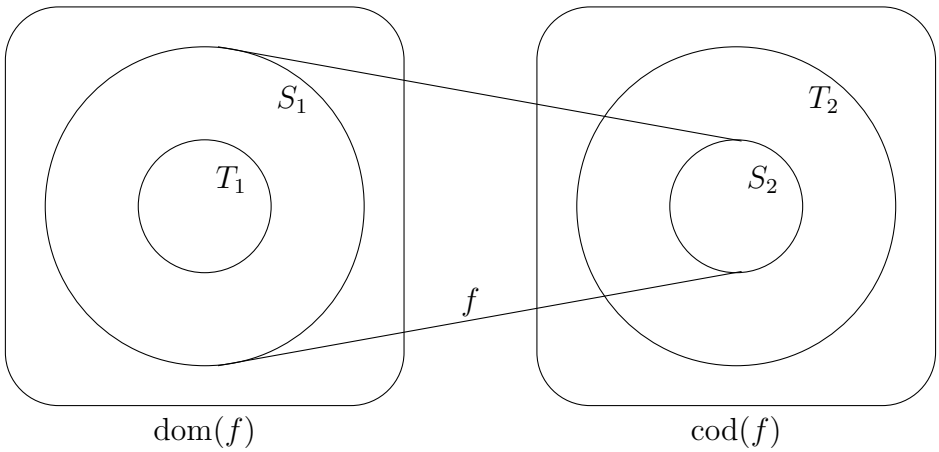
Infine dobbiamo specificare la regola di compatibilità per il costruttore di tipi "freccia":

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2} \, (\text{S}\to)$$

la regola si comporta in maniera *monotona* (o *covariante*) a destra delle frecce, mentre è *antimonotona* (o *controvariante*) a sinistra. Se pensassimo le funzioni come *grafi*, la regola che abbiamo appena scritto non sarebbe corretta, ad esempio:

$$\frac{\{0\} \subseteq \{0, 1\} \qquad \{2\} \subseteq \{2, 3\}}{\{< 0, 2 >, < 1, 2 >\} \not\subseteq \{< 0, 2 >, < 0, 3 >\}}$$

Invece la situazione che vogliamo modellare è la seguente:



cioè il tipo della funzione f che va da S_1 a S_2 è sottotipo del tipo delle funzioni che vanno da un dominio T_1 più piccolo ad un codominio T_2 più grande. In altre parole, una funzione f che va da S_1 a S_2 può sempre essere usata in contesti in cui ne serva una da T_1 a T_2: infatti ogni elemento $t_1 \in T_1$ è nel dominio di f, quindi $f(t_1)$ è sempre ben definita; inoltre $f(t_1)$ è sempre un elemento di T_2 (in particolare sarà sempre un elemento di S_2).

Ad esempio, vogliamo dimostrare che:

$$\vdash (\lambda x{:}\{n{:}\texttt{Nat}\}.\ \texttt{succ}(x.n))\ \{n{=}\underline{3},b{=}\texttt{tt}\} : \texttt{Nat}$$

per ragioni di spazio, spezziamo la dimostrazione in tre parti, chiamate rispettivamente Π_1:

$$\dfrac{\dfrac{\dfrac{}{x{:}\{n{:}\texttt{Nat}\} \vdash x{:}\{n{:}\texttt{Nat}\}}}{x{:}\{n{:}\texttt{Nat}\} \vdash x.n : \texttt{Nat}}}{x{:}\{n{:}\texttt{Nat}\} \vdash \texttt{succ}(x.n) : \texttt{Nat}}}{\vdash \lambda x{:}\{n{:}\texttt{Nat}\}.\ \texttt{succ}(x.n) : \{n{:}\texttt{Nat}\}{\to}\texttt{Nat}}$$

e Π_2:

$$\dfrac{\dfrac{\Pi_1}{\vdash \lambda x{:}\{n{:}\texttt{Nat}\}.\ \texttt{succ}(x.n) : \{n{:}\texttt{Nat}\}{\to}\texttt{Nat}} \qquad \dfrac{\{n{:}\texttt{Nat},b{:}\texttt{Bool}\} <: \{n{:}\texttt{Nat}\} \qquad \texttt{Nat} <: \texttt{Nat}}{\{n{:}\texttt{Nat}\}{\to}\texttt{Nat} <: \{n{:}\texttt{Nat},b{:}\texttt{Bool}\}{\to}\texttt{Nat}}}{\vdash \lambda x{:}\{n{:}\texttt{Nat}\}.\ \texttt{succ}(x.n) : \{n{:}\texttt{Nat},b{:}\texttt{Bool}\}{\to}\texttt{Nat}}$$

per concludere con Π_3:

$$\dfrac{\dfrac{\Pi_2}{\vdash \lambda x{:}\{n{:}\texttt{Nat}\}.\ \texttt{succ}(x.n) : \{n{:}\texttt{Nat},b{:}\texttt{Bool}\}{\to}\texttt{Nat}} \qquad \dfrac{\vdash \underline{3} : \texttt{Nat} \qquad \vdash \texttt{tt} : \texttt{Bool}}{\vdash \{n{=}\underline{3},b{=}\texttt{tt}\} : \{n{:}\texttt{Nat},b{:}\texttt{Bool}\}}}{\vdash (\lambda x{:}\{n{:}\texttt{Nat}\}.\ \texttt{succ}(x.n))\ \{n{=}\underline{3},b{=}\texttt{tt}\} : \texttt{Nat}}$$

Osserviamo che esiste un'altro modo di derivare il tipo della formula, riposizionando l'applicazione della regola di sussunzione, spezzando come sopra in Π'_2:

$$\dfrac{\dfrac{\vdash \underline{3} : \texttt{Nat} \qquad \vdash \texttt{tt} : \texttt{Bool}}{\vdash \{n{=}\underline{3},b{=}\texttt{tt}\} : \{n{:}\texttt{Nat},b{:}\texttt{Bool}\}} \qquad \{n{:}\texttt{Nat},b{:}\texttt{Bool}\} <: \{n{:}\texttt{Nat}\}}{\vdash \{n{=}\underline{3},b{=}\texttt{tt}\} : \{n{:}\texttt{Nat}\}}$$

per concludere con Π'_3:

$$\dfrac{\dfrac{\Pi_1}{\vdash \lambda x{:}\{n{:}\texttt{Nat}\}.\ \texttt{succ}(x.n) : \{n{:}\texttt{Nat}\}{\to}\texttt{Nat}} \qquad \dfrac{\Pi'_2}{\vdash \{n{=}\underline{3},b{=}\texttt{tt}\} : \{n{:}\texttt{Nat}\}}}{\vdash (\lambda x{:}\{n{:}\texttt{Nat}\}.\ \texttt{succ}(x.n))\ \{n{=}\underline{3},b{=}\texttt{tt}\} : \texttt{Nat}}$$

Osserviamo infine che le regole di transitività e di sussunzione hanno nelle premesse un termine (che abbiamo chiamato S in (Stran) e (Tsub)) che non compaiono nelle conclusioni, cioè dei *tagli*. Quindi la proprietà della sottoformula non è rispettata (nelle premesse appaiono appunto termini che *non sono* sottoformule delle conclusioni) e pertanto perdiamo le buone proprietà algoritmiche del sistema di tipi.

4.1 Proprietà

Lemma 4.1 (Inversione per sottotipi)**.**

(i) Se $S <: T_1 \to T_2$ *allora* $S \equiv S_1 \to S_2$ *e* $T_1 <: S_1$ *e* $S_2 <: T_2$*;*

(ii) Se $S <: \{l_1 : T_1, \ldots, l_n : T_n\}$ *allora:*

- $S \equiv \{k_1 : S_1, \ldots, k_m : S_m\}$*;*
- $\{l_i\} \subseteq \{k_j\}$*;*
- *per ogni* i, j *tali che* $l_i \equiv k_j$ *si ha* $S_j <: T_i$*.*

Dimostrazione. Per induzione sulla derivazione di sottotipo.

(i) Come abbiamo stabilito $S <: T_1 \to T_2$?

- Riflessività. Ovvio: $S \equiv T_1 \to T_2 \equiv S_1 \to S_2$.
- Regola (S$\to$).

$$\frac{T_1 <: S_1 \qquad S_2 <: T_2}{S <: T_1 \to T_2}$$

  Quindi $S \equiv S_1 \to S_2$ e $T_1 <: S_1$ e $S_2 <: T_2$.
- Transitività.

$$\frac{S <: R \qquad R <: T_1 \to T_2}{S <: T_1 \to T_2}$$

  per ipotesi induttiva sulla premessa di destra:

  1. $R \equiv R_1 \to R_2$;
  2. $T_1 <: R_1$;
  3. $R_2 <: T_2$;

  grazie a (1) sappiamo che possiamo applicare l'ipotesi induttiva sulla premessa di sinistra:

  4. $S \equiv S_1 \to S_2$;
  5. $R_1 <: S_1$;
  6. $S_2 <: R_2$;

  per transitività di (2) e (5), otteniamo $T_1 <: S_1$, mentre per transitività di (3) e (6) otteniamo $S_2 <: T_2$. Per (4) abbiamo $S \equiv S_1 \to S_2$.

(ii) Analogo al precedente.

□

Lemma 4.2 (Inversione per termini con sottotipi)**.**

(i) Se $\Gamma \vdash \lambda x : R.s_2 : T_1 \to T_2$ *allora* $T_1 <: R$ *e* $\Gamma, x : R \vdash s_2 : T_2$*;*

(ii) Se $\Gamma \vdash \{l_i{=}s_i\}_{i \in I} : \{k_j{=}T_j\}_{j \in J}$ *allora:*

- $\{k_j\}_{j \in J} \subseteq \{l_i\}_{i \in I}$*;*
- *per ogni* $i \in I, j \in J$ *tali che* $k_j \equiv l_i$ *si ha* $\Gamma \vdash s_i : T_j$*.*

Dimostrazione. Per induzione sul giudizio di tipo.

(i) L'unica regola con cui possiamo concludere il giudizio $\Gamma \vdash \lambda x : R.s_2 : T_1 \to T_2$ è sussunzione.

$$\frac{\Gamma \vdash \lambda x : R.s_2 : S \qquad S <: T_1 \to T_2}{\Gamma \vdash \lambda x : R.s_2 : T_1 \to T_2}$$

Per il lemma 4.2 (inversione per sottotipi), abbiamo, sulla premessa di destra:

1. $S \equiv S_1 \to S_2$;
2. $T_1 <: S_1$;
3. $S_2 <: T_2$;

mentre, per ipotesi induttiva sulla premessa di sinistra:

4. $S_1 <: R$;
5. $\Gamma, x : R \vdash s_2 : S_2$;

ora, per sussunzione applicata a (3) e (5), abbiamo $\Gamma, x : R \vdash s_2 : T_2$, e per transitività applicata a (2) e (4) $T_1 <: R$

(ii) Analogo.

□

Lemma 4.3 (Sostituzione)**.**

$$\textit{Se} \quad \Gamma, x : S \vdash t : T \quad \textit{e} \quad \Gamma \vdash s : S \quad \textit{allora} \quad \Gamma \vdash t[s/x] : T$$

Dimostrazione. Per induzione sulla derivazione.

□

Teorema 4.4 (Subject reduction)**.**

$$\textit{Se} \quad \Gamma \vdash t : T \quad \textit{e} \quad t \to t' \quad \textit{allora} \quad \Gamma \vdash t' : T$$

Dimostrazione. Per induzione sulla derivazione di $\Gamma \vdash t : T$.

- I casi per variabili e astrazioni sono vacuamente veri, perchè sono valori e quindi non riducono;
- Applicazione.

$$\frac{\Gamma \vdash t_1 : S{\to}T \qquad \Gamma \vdash t_2 : S}{\Gamma \vdash t_1 t_2 : T}$$

Dobbiamo considerare le possibili derivazioni di $t \to t'$:

(i) Riduciamo t_1:

$$\frac{t_1 \to t_1'}{t_1 t_2 \to t_1' t_2}$$

Per ipotesi induttiva $\Gamma \vdash t_1' : S{\to}T$, da cui:

$$\frac{\Gamma \vdash t_1' : S{\to}T \qquad \Gamma \vdash t_2 : S}{\Gamma \vdash t_1' t_2 : T}$$

(ii) $t_1 \equiv v_1$ è un valore, per cui riduciamo t_2:

$$\frac{t_2 \to t_2'}{v_1 t_2 \to v_1 t_2'}$$

Per ipotesi induttiva $\Gamma \vdash t_2' : S$, da cui:

$$\frac{\Gamma \vdash v_1 : S{\to}T \qquad \Gamma \vdash t_2' : S}{\Gamma \vdash v_1 t_2' : T}$$

(iii) $t_2 \equiv v_2$ è un valore e $t_1 \equiv \lambda x{:}R.s_1$ è una lambda astrazione, per cui la regola di riduzione da applicare è (Eβ):

$$(\lambda x{:}R.s_1)v_2 \to s_1[v_2/x]$$

Per il lemma 4.2 *di inversione* applicato a $\Gamma \vdash \lambda x{:}R.s_1$, abbiamo che $\Gamma, x : R \vdash s_1 : T$ e $S <: R$. Per sussunzione abbiamo:

$$\frac{\Gamma \vdash v_2 : S \qquad S <: R}{\Gamma \vdash v_2 : R}$$

e infine, da $\Gamma, x : R \vdash s_1 : T$ e $\Gamma \vdash v_2 : R$, per il lemma 4.3 *di sostituzione* otteniamo:

$$\Gamma \vdash s_1[v_2/x] : T$$

- Subsumption.

$$\frac{\Gamma \vdash t : S \qquad S <: T}{\Gamma \vdash t : T} \quad \text{e} \quad t \to t'$$

Per ipotesi induttiva sulla premessa di sinistra, $\Gamma \vdash t' : S$ e per sussunzione concludiamo $\Gamma \vdash t' : T$

□

Lemma 4.5 (Forme canoniche)**.**

- *Se* $\vdash v : T_1 \to T_2$ *allora* $v \equiv \lambda x : S_1.s_2$ *e* $T_1 <: S_1$*;*
- *Se* $\vdash v : \{l_1 : T_1, \ldots, l_n : T_n\}$ *allora:*
  - $v \equiv \{k_1 = v_1, \ldots, k_m = v_m\}$ *con* $v_1, \ldots, v_m$ *valori;*
  - $\{l_1, \ldots, l_n\} \subseteq \{k_1, \ldots, k_m\}$.

Dimostrazione. Immediata, via lemmi di inversione con sottotipi. □

Teorema 4.6 (Progresso)**.** *Sia* t *chiuso e ben tipizzato* ($\vdash t : T$)*. Allora* t *è un valore oppure* $t \to t'$ *per qualche* t'*.*

Dimostrazione. Per induzione sulla derivazione di $\vdash t : T$ usando il lemma 4.5 di *forme canoniche.*

□

Consideriamo le regole per:

- *Cast*:

$$\frac{\Gamma \vdash t : T \qquad T <: S}{\Gamma \vdash t \ \texttt{as} \ S : S} \ (\text{T}_{\text{CAST}^{\text{up}}}) \qquad \frac{\Gamma \vdash t : S \qquad T <: S}{\Gamma \vdash t \ \texttt{as} \ T : T} \ (\text{T}_{\text{CAST}^{\text{down}}})$$

  Per utilizzare queste regole è necessario introdurre un controllo dinamico di tipo, da effettuarsi durante la riduzione, ed utilizzare la regola:

$$\frac{\vdash v : T}{v \ \texttt{as} \ T \to v} \ (\text{E}_{\text{CAST}^{\text{dyn}}})$$

- *Varianti*: in maniera del tutto analoga a quanto fatto per i record, abbiamo tre nuove regole di tipo per i varianti: *compatibilità in ampiezza*, *in profondità* e *permutazione.*

- *Riferimenti*: non è semanticamente corretto assumere qualche relazione tra $\texttt{Ref}\ T$ e $\texttt{Ref}\ S$, anche nel caso in cui $S <: T$ oppure $T <: S$. Consideriamo la regola "ingenua":

$$\frac{T <: S}{\texttt{Ref}\ T <: \texttt{Ref}\ S}\ (\textsc{Sref})$$

in lettura non crea problemi, però la stessa regola non vale per l'accesso in scrittura. Ad esempio, siano:

$$\begin{array}{rcl} T & \equiv & \{n_1\texttt{:Nat},n_2\texttt{:Nat}\} \\ S & \equiv & \{n_1\texttt{:Nat}\} \\ t & \triangleq & \texttt{ref}\ \{n_1\texttt{=}\underline{1},n_2\texttt{=}\underline{2}\} \end{array}$$

È possibile derivare:

$$\dfrac{\vdash \{n_1\texttt{=}\underline{7}\} : S \qquad \dfrac{\vdash t : \texttt{Ref}\ T \qquad \dfrac{T <: S}{\texttt{Ref}\ T <: \texttt{Ref}\ S}}{\vdash t : \texttt{Ref}\ S}}{\vdash t\texttt{:=}\{n_1\texttt{=}\underline{7}\} : \texttt{Unit}}$$

Ma questo sistema vìola la correttezza, perchè ora leggendo t:

$$\vdash\ !t : T$$

mentre:

$$\not\vdash \{n_1\texttt{=}\underline{7}\} : T$$

Pertanto:

$$\texttt{Ref}\ T \not<: \texttt{Ref}\ S$$

L'unico modo per aggirare questo problema è aggiungere dei controlli dinamici di tipo, come avviene ad esempio per gli array nel linguaggio Java. Infatti in Java esiste una regola del tipo:

$$\frac{S <: T}{S\texttt{[]} <: T\texttt{[]}}$$

Il compilatore Java accetta come ben tipizzato il seguente programma, ma quando eseguito genera un errore di tipo a *runtime* in corrispondenza della riga 5.

```
1  class Test {
2     public static void main(String[] args) {
3        String[] w = new String[3];
4        Object[] v = w;
5        v[0] = new Object();
6        if(w[0].isEmpty())
7           System.out.println("What?!");
8     }
9  }
```

4.1.1 Un sistema ristretto

Ora vorremmo un algoritmo per decidere se un tipo è sottotipo di un altro. Per tornare ad un sistema che gode della proprietà della sottoformula, e quindi anche di buone proprietà algoritmiche, dobbiamo apportare alcune modifiche. Transitività e riflessività sono regole *eliminabili*, mentre la subsumption può essere modificata per adattarla solo a casi più specifici.

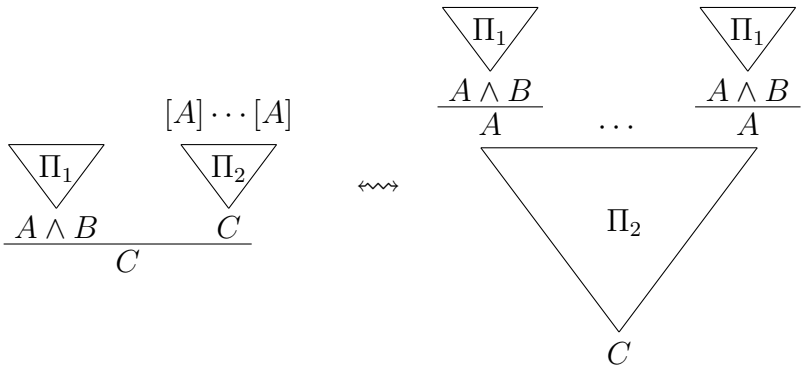
Per illustrare la differenza fra *regoli derivabili* e *regole eliminabili* prendiamo ad esempio il sistema logico proposizionale. Scriviamo la regola di *eliminazione sinistra della congiunzione* e la regola di *eliminazione della congiunzione generalizzata*:

$$\frac{A \wedge B}{A}\,\mathcal{E}\wedge \qquad \frac{A \wedge B \qquad \begin{array}{c}[A]\\ \cdots \\ C\end{array}}{C}\,\mathcal{E}\wedge g$$

- $\mathcal{E}\wedge$ è *derivabile* da $\mathcal{E}\wedge g$:

$$\frac{A \wedge B \qquad [A]}{A} \quad \equiv \quad \frac{A \wedge B}{A}$$

- $\mathcal{E}\wedge g$ è *eliminabile* in presenza di $\mathcal{E}\wedge$; questo significa che la prova va ristrutturata:



Consideriamo il sistema di tipi con `Top`, freccia, record, tipi base: `Bool`, `Nat`, ...

Teorema 4.7. *Supponiamo di avere una derivazione di $S <: T$. Allora esiste una derivazione della stessa conclusione che non usa riflessività (eccetto che sui tipi primitivi* `Nat`*,* `Bool`*, ...).*

La dimostrazione di questo teorema segue direttamente da:

Lemma 4.8. *Per ogni tipo T, $T <: T$ è derivabile senza far uso di riflessività, eccetto che sui tipi primitivi.*

Dimostrazione. Per induzione su T:

- $T \equiv \texttt{Nat}, T \equiv \texttt{Bool}$. Ovvio;
- $T \equiv \texttt{Top}$. In questo caso usiamo la regola:

  $$S <: \texttt{Top} \quad \text{per ogni tipo } S$$

  in particolare quando $S \equiv \texttt{Top}$ abbiamo $\texttt{Top} <: \texttt{Top}$;
- gli altri casi procedono per induzione immediata.

□

Teorema 4.9. *Se $\vdash S <: T$ allora esiste una derivazione di questo giudizio che non usa transitività.*

Dimostrazione. Sia:

$$\begin{array}{c} \Pi \\ \hline S <: T \end{array}$$

che usa transitività. Procediamo per induzione sull'altezza di Π. Tutti i casi sono immediati per induzione, eccetto quando l'ultima regola applicata è trasitività:

$$\dfrac{\begin{array}{cc} \Pi_1 & \Pi_2 \\ \hline S <: U & U <: T \end{array}}{S <: T}$$

Procediamo per casi sulle conclusioni di Π_1 e Π_2:

- Se Π_2 termina con (Stop) allora concludiamo subito $S <: \texttt{Top}$;
- Se Π_1 termina con (Stop), cioè $S <: \texttt{Top}$ e $\texttt{Top} <: T$, l'unica regola applicabile a destra è $\texttt{Top} <: \texttt{Top}$, per cui possiamo concludere direttamente $S <: \texttt{Top}$;

- Π_1 e Π_2 terminano entrambi con (S→):

$$\dfrac{\dfrac{\overset{\Pi_{11}}{\nabla}\quad U_1 <: S_1 \qquad \overset{\Pi_{12}}{\nabla}\quad S_2 <: U_2}{S_1 \to S_2 <: U_1 \to U_2} \qquad \dfrac{\overset{\Pi_{21}}{\nabla}\quad T_1 <: U_1 \qquad \overset{\Pi_{22}}{\nabla}\quad U_2 <: T_2}{U_1 \to U_2 <: T_1 \to T_2}}{S_1 \to S_2 <: T_1 \to T_2}$$

  Ristrutturiamo la prova sapendo che:

$$\dfrac{T_1 <: U_1 \qquad U_1 <: S_1}{T_1 <: S_1}$$

  è riformulabile, per ipotesi induttiva, in:

$$\overset{\Pi_3}{\nabla}\quad T_1 <: S_1$$

  che non usa transitività. Analogamente:

$$\dfrac{S_2 <: U_2 \qquad U_2 <: T_2}{S_2 <: T_2} \quad \overset{(IH)}{\longmapsto} \quad \overset{\Pi_4}{\nabla}\quad S_2 <: T_2$$

  con Π_4 che non usa transitività. Pertanto la nuova prova diventa:

$$\dfrac{\overset{\Pi_3}{\nabla}\quad T_1 <: S_1 \qquad \overset{\Pi_4}{\nabla}\quad S_2 <: T_2}{S_1 \to S_2 <: T_1 \to T_2}$$

□

Il "trucco" sta nello spingere le applicazioni della regola di transitività verso le foglie finchè non diventano `Top` o l'identità.

Dal fatto che (Srefl) e (Stran) sono eliminabili, discende l'esistenza di un semplice algoritmo che procede per induzione sulla struttura dei tipi che testa se $S <: T$. Concentriamoci ora sulla regola di subsumption; ad esempio, siano:

$$\begin{aligned} t_1 &\triangleq \lambda x{:}\{n\texttt{:Nat}\}.\, x.n \\ t_2 &\triangleq \{n\texttt{=}\underline{5}\texttt{,}m\texttt{=}\underline{7}\} \end{aligned}$$

Si può procedere in due modi per tipizzare il termine $(t_1 t_2)$:

1. Sussunzione sul funzionale:

$$\dfrac{\dfrac{\dfrac{x{:}\{n{:}\mathtt{Nat}\} \vdash x{:}\{n{:}\mathtt{Nat}\}}{x{:}\{n{:}\mathtt{Nat}\} \vdash x.n : \mathtt{Nat}}}{\vdash t_1 : \{n{:}\mathtt{Nat}\}{\to}\mathtt{Nat}} \qquad \dfrac{\cdots}{\{n{:}\mathtt{Nat}\}{\to}\mathtt{Nat} <: \{n{:}\mathtt{Nat}, m{:}\mathtt{Nat}\}{\to}\mathtt{Nat}}}{\vdash t_1 : \{n{:}\mathtt{Nat}, m{:}\mathtt{Nat}\}{\to}\mathtt{Nat}} \qquad \cdots}{\vdash t_1 t_2 : \mathtt{Nat}}$$

2. Sussunzione sull'argomento:

$$\dfrac{\vdash t_1 : \{n{:}\mathtt{Nat}\}{\to}\mathtt{Nat} \qquad \dfrac{\dfrac{\vdash \underline{5} : \mathtt{Nat} \qquad \vdash \underline{7} : \mathtt{Nat}}{\vdash t_2 : \{n{:}\mathtt{Nat}, m{:}\mathtt{Nat}\}} \qquad \{n{:}\mathtt{Nat}, m{:}\mathtt{Nat}\} <: \{n{:}\mathtt{Nat}\}}{\vdash t_2 : \{n{:}\mathtt{Nat}\}}}{\vdash t_1 t_2 : \mathtt{Nat}}$$

Vorremmo eliminare la possibilità di effettuare derivazioni del tipo (1), cioè quelle che applicano sussunzione sulle funzioni.

Definizione 4.10. *Sia* $\vdash$ *il sistema con sottotipi canonico. Chiamiamo* $\vdash_r$ *il* sistema ristretto *in cui abbiamo tolto subsumption e la regola dell'applicazione, ed abbiamo aggiunto al loro posto:*

$$\dfrac{\Gamma \vdash_r t : S{\to}T \qquad \Gamma \vdash_r s : R \qquad R <: S}{\Gamma \vdash_r t\, s : T}\ (\textsc{Tsub}^r)$$

Teorema 4.11.

$$\textit{Se} \quad \Gamma \vdash_r t : T \quad \textit{allora} \quad \Gamma \vdash t : T$$

Dimostrazione. Ovvio, la nuova regola è derivabile in $\vdash$. □

Teorema 4.12.

$$\textit{Se} \quad \Gamma \vdash t : T \quad \textit{allora} \quad \Gamma \vdash_r t : R \quad e \quad R <: T$$

Dimostrazione. Per induzione sulla derivazione di $\Gamma \vdash t : T$. I casi sono tutti immediati per ipotesi induttiva, a meno di:

- Applicazione.

$$\dfrac{\Gamma \vdash t_1 : T{\to}S \qquad \Gamma \vdash t_2 : T}{\Gamma \vdash t_1 t_2 : S}$$

  Per ipotesi induttiva, abbiamo:

(i) $\Gamma \vdash_r t_1 : R$ e $R <: T \to S$;

(ii) $\Gamma \vdash_r t_2 : Q$ e $Q <: T$;

e per il lemma 4.2 *d'inversione*:

(iii) $R \equiv R_1 \to R_2$;

(iv) $T <: R_1$;

(v) $R_2 <: S$;

Per transitività della relazione di sottotipo, se $Q <: T$ e $T <: R_1$ allora $Q <: R_1$, da cui:

$$\frac{\Gamma \vdash_r t_1 : R_1 \to R_2 \qquad \Gamma \vdash_r t_2 : Q \qquad Q <: R_1}{\Gamma \vdash_r t_1 t_2 : R_2}$$

- Sussunzione.

$$\frac{\Gamma \vdash t : S \qquad S <: T}{\Gamma \vdash t : T}$$

Per ipotesi induttiva $\Gamma \vdash_r t : R$ e $R <: S$, da cui, per transitività della relazione di sottotipo, $R <: T$.

□

Per concludere, abbiamo visto come le regole di riflessività e transitività siano superflue, perchè la relazione di sottotipo ne gode intrinsecamente; inoltre il sistema ristretto che abbiamo ottenuto modificando la regola di subsumption conserva tutta la capacità espressiva di quello canonico, fornendoci però una formulazione priva di tagli e quindi più adatta ad un uso algoritmico.

4.2 Tipi ricorsivi

Partiamo da un esempio pratico: la lista di naturali.

$$\begin{array}{rcl} \text{NatList} & = & \texttt{Unit} \;+\; \texttt{Nat} \times \text{NatList} \\ \text{nil} & \triangleq & \texttt{inl} * \;\texttt{as}\; \text{NatList} \\ \text{cons} & \triangleq & \lambda n{:}\texttt{Nat}.\lambda l{:}\text{NatList}.\; \texttt{inr}\; (n,l)\; \texttt{as}\; \text{NatList} \\ \text{head} & \triangleq & \lambda l{:}\text{NatList}.\; \texttt{case}\; l\; \texttt{of}\; \texttt{inl}\, x \Rightarrow \underline{0}\; \texttt{|}\; \texttt{inr}\, x \Rightarrow \texttt{fst}\, x \end{array}$$

Usiamo la seguente notazione:

$$\text{NatList} = \mu X.\texttt{Unit} + \texttt{Nat}\times X$$

dove μ, analogamente a λ, è un *binder* che lega *variabili di tipo* anzichè termini. Come vedremo in seguito, μ indica anche un punto fisso.

Consideriamo un altro esempio, lo *stream di naturali*:

$$\text{NatStream} = \mu X.\texttt{Unit}\to\texttt{Nat}\times X$$

La notazione funzionale che ha come argomento `Unit` serve a "bloccare" la computazione (poichè siamo in call-by-value) finchè non si riceve il prossimo valore $*$.

Il tipo NatStream corrisponde intuitivamente ad una lista infinita, a cui forniamo $*$ per accedere all'elemento successivo.

$$\begin{array}{rcl} \text{zeri} & \triangleq & \texttt{fix}\,(\lambda f{:}\text{NatStream}.\ \lambda u{:}\texttt{Unit}.\ (\texttt{0}, f)) \\ \text{naturali-da} & \triangleq & \texttt{fix}\,(\lambda f{:}\texttt{Nat}\to\text{NatStream}.\lambda n{:}\texttt{Nat}.\lambda u{:}\texttt{Unit}.\ (n, f\ (\texttt{succ}\ n))) \\ \text{naturali} & \triangleq & (\underline{\text{naturali-da}}\ \texttt{0}) \end{array}$$

Osserviamo l'operatore di punto fisso Y per il tipo T:

$$Y_T \triangleq \lambda f : T\to T.(\lambda x : \_.f(xx))(\lambda x : \_.f(xx))$$

Normalmente, per il lemma d'inversione, è impossibile che valgano assieme:

- $\vdash x : T$;
- $\vdash x : T\to T$;

ma questo non è più un problema in presenza di tipi ricorsivi. Infatti:

$$Y_T \triangleq \lambda f : T\to T.(\lambda x : \mu X.X\to T.f(xx))(\lambda x : \mu X.X\to T.f(xx))$$

è tipizzabile. In presenza di tipi ricorsivi, ogni tipo è abitato. Consideriamo il tipo:

$$\mu X.X\to X$$

la sua equazione ricorsiva associata è:

$$\Lambda = \Lambda\to\Lambda$$

I termini di tipo Λ sono isomorfi ai termini del lambda calcolo senza tipi.

Definizione 4.13. *L'insieme degli* alberi infiniti, etichettati e con radice *coincide con le funzioni parziali:*

$$\mathscr{T} : \{1,2\}^* \to \{\mathtt{Top}, \times, \to\}$$

tali che, per ogni $T \in \mathscr{T}$*:*

- $T(\epsilon)$ *è definito (cioè l'albero ha radice);*
- *se* $T(\pi\sigma)$ *è definito, allora anche* $T(\pi)$ *lo è.*

Il linguaggio generato dalla grammatica dei tipi:

$$T ::= \mathtt{Top} \mid T{\times}T \mid T{\to}T$$

può essere visto, in maniera denotazionale, come punto fisso di un operatore (su alberi):

$$S(X) = \{\mathtt{Top}\} \cup \{u{\times}v \mid u \in X, v \in X\} \cup \{u{\to}v \mid u \in X, v \in X\}$$

mentre la relazione di sottotipo:

$$\frac{}{T <: \mathtt{Top}} \qquad \frac{S_1 <: T_1 \qquad S_2 <: T_2}{S_1{\times}S_2 <: T_1{\times}T_2} \qquad \frac{T_1 <: S_1 \qquad S_2 <: T_2}{S_1{\to}S_2 <: T_1{\to}T_2}$$

si può vedere come punto fisso di un operatore S_t così definito:

$$\begin{aligned} S_t \quad &: \quad \mathscr{P}(\mathscr{T}{\times}\mathscr{T}) \to \mathscr{P}(\mathscr{T}{\times}\mathscr{T}) \\ S_t(R) \quad &= \quad \{(T, \mathtt{Top}) \mid T{\in}\mathscr{T}\} \cup \{(S_1{\times}S_2, T_1{\times}T_2) \mid (S_1,T_1) \in R, (S_2,T_2) \in R\} \cup \\ &\qquad \cup\{(S_1{\to}S_2, T_1{\to}T_2) \mid (T_1,S_1) \in R, (S_2,T_2) \in R\} \end{aligned}$$

Teorema 4.14 (Knaster–Tarski)**.** *Se* $F : \mathscr{P}(\mathcal{U}) \to \mathscr{P}(\mathcal{U})$ *è continua, cioè:*

$$F(\bigcup_{i\in\mathbb{N}} X_i) = \bigcup_{i\in\mathbb{N}} F(X_i)$$

per ogni catena crescente $\{X_i\}_{i\in\mathbb{N}}$*, allora:*

$$\mathrm{lfp}(F) = \bigcup_{i\in\mathbb{N}} F^i(\emptyset)$$

Dualmente, se $F : \mathscr{P}(\mathcal{U}) \to \mathscr{P}(\mathcal{U})$ *è co-continua, cioè:*

$$F(\bigcap_{i\in\mathbb{N}} X_i) = \bigcap_{i\in\mathbb{N}} F(X_i)$$

per ogni catena decrescente $\{X_i\}_{i\in\mathbb{N}}$*, allora:*

$$\mathrm{gfp}(F) = \bigcap_{i\in\mathbb{N}} F^i(\mathcal{U})$$

Se interpretiamo le regole per la relazione $<:$ di sottotipo in maniera coinduttiva, stiamo cercando il massimo punto fisso dell'operatore S_t. Adottiamo la seguente notazione:

$$\begin{array}{rcl} \mu F & \triangleq & \mathrm{lfp}(F) \\ \nu F & \triangleq & \mathrm{gfp}(F) \end{array}$$

Definizione 4.15. *Il* principio di induzione *ci permette di ragionare sul* lfp*:*

$$\frac{F(X) \subseteq X}{\mu F \subseteq X}$$

L'induzione sui naturali è un caso particolare di questa regola. Infatti, dalla seguente definizione grammaticale:

$$n ::= \mathtt{0} \mid \mathtt{s}\ n$$

si ottiene che, come minimo punto fisso dell'operatore:

$$F(X) = \{\mathtt{0}\} \cup \{\mathtt{s}\ n \mid n \in X\}$$

abbiamo un insieme isomorfo ai naturali $\mu F \approx \mathbb{N}$. L'usuale formulazione del principio di induzione:

$$\frac{P(\mathtt{0}) \qquad P(n) \rightarrow P(\mathtt{s}\ n)}{\forall n.P(n)}$$

diventa:

$$\frac{F(P) \subseteq P}{\mathbb{N} \subseteq P} \quad \text{cioè} \quad \frac{\{\mathtt{0}\} \subseteq P \qquad \{\mathtt{s}\ n \mid n \in P\} \subseteq P}{\mathbb{N} \subseteq P}$$

e chiaramente se $\mathbb{N} \subseteq P$, allora P vale su tutti i naturali. Dualmente:

Definizione 4.16. *Il* principio di coinduzione *ci permette di ragionare sul* gfp*:*

$$\frac{X \subseteq F(X)}{X \subseteq \nu F}$$

Cioè si può usare il *principio di coinduzione* per dimostrare che per un certo $\overline{x}$ vale $\overline{x} \in \nu F$. Prendiamo ad esempio i tipi:

$$\begin{array}{rcl} S_0 & \equiv & \mu X.\mathtt{Top} \times X \\ T_0 & \equiv & \mu X.\mathtt{Top} \times (\mathtt{Top} \times X) \end{array}$$

vogliamo dimostrare che $(S_0, T_0) \in \nu S_t$ (cioè $S_0 <: T_0$). Definiamo R come segue:

$$R \triangleq \{(S_0, T_0), (\texttt{Top}, \texttt{Top}), (\texttt{Top}\times S_0, \texttt{Top}\times T_0)\}$$

e calcoliamo $S_t(R)$:

$$S_t(R) \quad = \quad \{(T, \texttt{Top}) \mid T \in \mathscr{T}\} \cup \{(\texttt{Top}\times S_0, \texttt{Top}\times T_0), (\texttt{Top}\times S_0, \texttt{Top}\times(\texttt{Top}\times T_0))\} \cup \\ \cup \ldots$$

Siccome $R \subseteq S_t(R)$, possiamo applicare il principio di coinduzione:

$$\frac{R \subseteq S_t(R)}{R \subseteq \nu S_t}$$

e pertanto $(S_0, T_0) \in \nu S_t$.

Per un altro semplice esempio, vedi Pierce [2002] (pagine 291-292).

Segue un possibile algoritmo di subtyping, che verifica, dati due tipi S e T se $S <: T$, cioè se $(S, T) \in \nu S_t$. L'algoritmo restituisce un insieme di ipotesi oppure `fail`.

Figura 2 Algoritmo di subtyping

$\mathrm{sub}(\Gamma, S, T) \triangleq$
  `if` $(S, T) \in \Gamma$ `then` Γ
    `else let` $\Gamma_0 = \Gamma \cup \{(S, T)\}$ `in`
      `if` $T \equiv$ `Top then` Γ_0
      `else if` $S \equiv S_1\times S_2$ `and` $T \equiv T_1\times T_2$ `then`
        `let` $\Gamma_1 = \mathrm{sub}(\Gamma_0, S_1, T_1)$ `in` $\mathrm{sub}(\Gamma_1, S_2, T_2)$
      `else if` $S \equiv S_1\to S_2$ `and` $T \equiv T_1\to T_2$ `then`
        `let` $\Gamma_1 = \mathrm{sub}(\Gamma_0, T_1, S_1)$ `in` $\mathrm{sub}(\Gamma_1, S_2, T_2)$
      `else if` $T \equiv \mu X.T_1$ `then`
        $\mathrm{sub}(\Gamma_0, S, T_1[T/X])$
      `else if` $S \equiv \mu X.S_1$ `then`
        $\mathrm{sub}(\Gamma_0, S_1[S/X], T)$
      `else fail`.

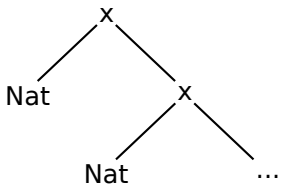
Vorremmo dimostrare che l'algoritmo di subtyping è:

- corretto, cioè che se $\mathrm{sub}(\emptyset, S, T) \neq$ `fail` allora $(S, T) \in \nu S_t$;
- completo, cioè che se $(S, T) \in \nu S_t$ allora $\mathrm{sub}(\emptyset, S, T) \neq$ `fail`;

- totale: ma questo fatto è *falso*, come vedremo nell'esempio seguente.

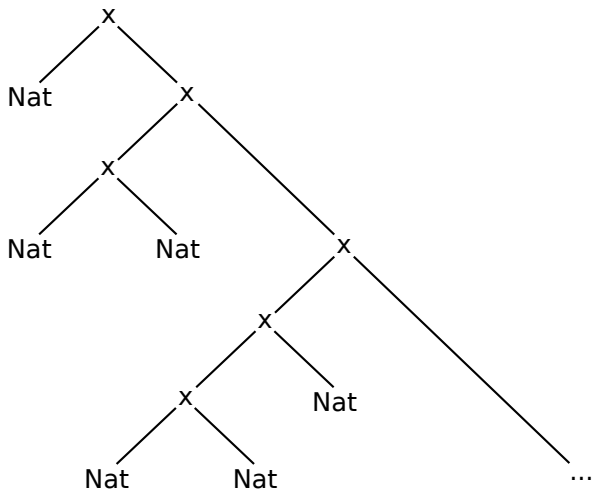
Consideriamo i seguenti tipi:

a)



$$\mu X.\texttt{Nat}\times X$$

b)



$$\begin{aligned}
&\texttt{Nat} &&\times\\
&(\texttt{Nat}\times\texttt{Nat}) &&\times\\
&((\texttt{Nat}\times\texttt{Nat})\times\texttt{Nat}) &&\times\\
&(((\texttt{Nat}\times\texttt{Nat})\times\texttt{Nat})\times\texttt{Nat}) &&\times\\
&\ldots
\end{aligned}$$

In questo caso l'algoritmo di subtyping non termina. Tuttavia non riusciamo ad esprimere il tipo per il secondo albero, nonostante quest'ultimo appartenga a $\mathscr{T}$. Questo ci spinge a restringere il dominio degli alberi, a considerare cioè i cosiddetti "alberi regolari".

Definizione 4.17 (Sottoalbero)**.** *Un albero $S \in \mathscr{T}$ è un* sottoalbero *di $T \in \mathscr{T}$ se:*

$$S = \lambda\sigma.T(\pi\sigma)$$

per qualche π, cioè S si ottiene dall'albero T percorrendo un certo cammino π che parte dalla radice e prendendo il sottoalbero ivi radicato.

Definizione 4.18 (Alberi regolari)**.** *Un albero in $\mathscr{T}$ è* regolare *se l'insieme dei suoi sottoalberi è finito.*

Definizione 4.19 (Pre-μ-tipi contrattivi)**.** *Si definiscono* pre-μ-tipi contrattivi *(da qui in avanti* μ-tipi*) i tipi generati dalla seguente grammatica:*

$$\mathscr{T}_m ::= \mathtt{Top} \mid X \mid \mathscr{T}_m \times \mathscr{T}_m \mid \mathscr{T}_m \to \mathscr{T}_m \mid \mu X.\mathscr{T}_m$$

in cui vale che, in ogni sottoespressione, il cammino da un binder alla variabile che lega è di lunghezza strettamente maggiore di 1.

Osserviamo che gli alberi infiniti generati da queste espressioni hanno tutti un numero finito di sottoalberi differenti, cioè corrispondono agli *alberi regolari*.

Ridefiniamo l'operatore S_t che identifica la relazione di sottotipo come massimo punto fisso:

Definizione 4.20 (Operatore di sottotipo S_t)**.**

$$\begin{aligned} S_t \quad &: \quad \mathscr{P}(\mathscr{T}_m \times \mathscr{T}_m) \to \mathscr{P}(\mathscr{T}_m \times \mathscr{T}_m) \\ S_t(R) \quad &\triangleq \quad \{(T, \mathtt{Top}) \mid T \in \mathscr{T}_m\} \cup \\ &\qquad \cup\{(S_1 \times S_2, T_1 \times T_2) \mid (S_1, T_1) \in R, (S_2, T_2) \in R\} \cup \\ &\qquad \cup\{(S_1 \to S_2, T_1 \to T_2) \mid (T_1, S_1) \in R, (S_2, T_2) \in R\} \cup \\ &\qquad \cup\{(S, \mu X.T) \mid (S, T[\mu X.T/X]) \in R\} \cup \\ &\qquad \cup\{(\mu X.S, T) \mid (S[\mu X.S/X], T) \in R, T \not\equiv \mathtt{Top}, T \not\equiv \mu Y.T_1\} \end{aligned}$$

Ci chiediamo se la regola di transitività sia necessaria. Definiamo un nuovo operatore:

$$tr(X) \triangleq \{(a, b) \mid \exists c \in \mathscr{T}_m.(a, c) \in X, (c, b) \in X\}$$

ed aggiungiamo la transitività alle regole precedenti, ottenendo:

$$S_{tr}(R) \triangleq S_t(R) \cup tr(R)$$

ora vale che:

$$\nu S_{tr} = \mathscr{T}_m \times \mathscr{T}_m$$

Dimostrazione. Per dimostrare questo fatto (vale per ogni operatore monotono), basta provare che $\mathscr{T}_m \times \mathscr{T}_m \subseteq \nu S_{tr}$ e per coinduzione basta provare $\mathscr{T}_m \times \mathscr{T}_m \subseteq S_{tr}(\mathscr{T}_m \times \mathscr{T}_m)$. Sia $(T, S) \in \mathscr{T}_m \times \mathscr{T}_m$; per ogni $R \in \mathscr{T}_m$, vale anche $(T, R) \in \mathscr{T}_m \times \mathscr{T}_m$ e $(R, S) \in \mathscr{T}_m \times \mathscr{T}_m$, da cui:

$$(T, S) \in tr(\mathscr{T}_m \times \mathscr{T}_m) \subseteq S_{tr}(\mathscr{T}_m \times \mathscr{T}_m)$$

□

Lemma 4.21. *Sia* $F : \mathscr{P}(\mathscr{T}_m \times \mathscr{T}_m) \to \mathscr{P}(\mathscr{T}_m \times \mathscr{T}_m)$ *monotono.*

$$Se \qquad \forall R \in \mathscr{T}.tr(F(R)) \subseteq F(tr(R)) \qquad allora \qquad \nu F \text{ } è \text{ } transitiva$$

Dimostrazione. Per la proprietà fondamentale dei punti fissi $tr(\nu F) = tr(F(\nu F))$ e per ipotesi $tr(F(\nu F)) \subseteq F(tr(F(\nu F)))$. Ora, per coinduzione:

$$\frac{tr(\nu F) \subseteq F(tr(\nu F))}{tr(\nu F) \subseteq \nu F}$$

cioè νF è transitiva. □

Teorema 4.22. νS_t *è transitiva.*

Dimostrazione. In virtù del *lemma 4.21*, basta dimostrare che:

$$\forall R \in \mathscr{T}_m.tr(S_t(R)) \subseteq S_t(tr(R))$$

Sia $(A, B) \in tr(S_t(R))$, cioè $\exists C \in \mathscr{T}_m.(A, C) \in S_t(R)$ e $(C, B) \in S_t(R)$. Per casi su C:

- $C \equiv \texttt{Top}$. Se $(\texttt{Top}, B) \in S_t(R)$ allora $B \equiv \texttt{Top}$ e quindi $(A, \texttt{Top}) \in S_t(tr(R))$.
- $C \equiv C_1 \times C_2$. Per casi su B:
  - $B \equiv \texttt{Top}$, come prima.
  - Se $B \not\equiv \texttt{Top}$ allora $B \equiv B_1 \times B_2$ e $(C_2, B_2) \in R$ e allora necessariamente $A \equiv A_1 \times A_2$ e $(A_1, C_1) \in R$, $(A_2, C_2) \in R$. Da cui $(A_1, B_1) \in tr(R)$, $(A_2, B_2) \in tr(R)$. Quindi $(A_1 \times A_2, B_1 \times B_2) \equiv (A, B) \in S_t(tr(R))$.
  - $C \equiv C_1 \to C_2$. Analogo.

□

Questo teorema ci dice che la transitività è una regola *ammissibile* nel calcolo, cioè che non è necessaria perchè νS_t è già transitiva. Si dimostra in maniera analoga (anche più semplice) che anche la riflessività è una regola ammissibile.

Capitolo 5

Polimorfismo parametrico esplicito

Vogliamo estendere la sintassi del calcolo per inglobare il concetto di polimorfismo a livello dei termini. Il risultato che vogliamo ottenere è, ad esempio, passare da un termine nella forma:

$$\lambda f : T{\to}T.\lambda g : T{\to}T.\lambda x : T.(f(gx)) : (T{\to}T){\to}(T{\to}T){\to}T{\to}T$$

in cui il comportamento polimorfo è implicito, a:

$$\Lambda T.\lambda f : T{\to}T.\lambda g : T{\to}T.\lambda x : T.(f(gx)) : \forall T.(T{\to}T){\to}(T{\to}T){\to}T{\to}T$$

Il calcolo che otteniamo è noto in letteratura come *Sistema F*:

[M] $t ::= x \mid \lambda x : T.t \mid t\, t \mid \Lambda X.t \mid t\, T$

[V] $v ::= \lambda x : T.t \mid \Lambda X.t$

[E] Sia v un valore:

$$(\Lambda X.t)T \to t[T/X] \quad (\mathrm{E_F})$$

$$\frac{t \to t'}{t\, T \to t'\, T}\,(\mathrm{E_F^{ctx}})$$

$$(\lambda x : T.t)v \to t[v/x] \quad (\mathrm{E}\beta)$$

$$\frac{t_1 \to t_1'}{t_1\, t_2 \to t_1'\, t_2}\,(\mathrm{E}\beta^{\mathrm{LX}}) \qquad \frac{t_2 \to t_2'}{v\, t_2 \to v\, t_2'}\,(\mathrm{E}\beta^{\mathrm{RX}})$$

[S] $T ::= X \mid T{\to}T \mid \forall X.T$

`[ T ]` Oltre alle regole di tipizzazione viste in precedenza, aggiungiamo:

$$\frac{\Gamma \vdash t : \forall X.T}{\Gamma \vdash t\ S : T[S/X]}\ (\mathrm{T}\forall^{\mathrm{elim}}) \qquad \frac{\Gamma \vdash t : T \qquad X \notin FV(\Gamma)}{\Gamma \vdash \Lambda X.t : \forall X.T}\ (\mathrm{T}\forall^{\mathrm{intro}})$$

Seguono alcuni esempi di termini appartenenti al sistema F. Osserviamo che questo termine:

$$\lambda x : \forall X.X{\to}X.((x(\forall X.X{\to}X))x) : (\forall X.X{\to}X){\to}(\forall X.X{\to}X)$$

è una forma di autoapplicazione. Tuttavia si dimostra che il sistema F è fortemente normalizzante.

Il termine:

$$\Lambda X.\lambda x : X.\lambda y : X.x : \forall X.X{\to}X{\to}X$$

corrisponde ad una versione polimorfa del *combinatore* **K**. Diamo un nome al tipo di questo termine:

$$\mathrm{CBool} \triangleq \forall X.X{\to}X{\to}X$$

Intuitivamente la prima occorrenza della X corrisponde al termine booleano `tt`, la seconda a `ff`, e la terza è il risultato. Ad esempio:

$$\begin{array}{rcl}
\mathrm{tt} & \triangleq & \Lambda X.\lambda x : X.\lambda y : X.x : \underline{\mathrm{CBool}} \\
\mathrm{ff} & \triangleq & \Lambda X.\lambda x : X.\lambda y : X.y : \underline{\mathrm{CBool}} \\
\mathrm{not} & \triangleq & \lambda a : \underline{\mathrm{CBool}}.(a\ \underline{\mathrm{CBool}}\ \underline{\mathrm{ff}}\ \underline{\mathrm{tt}}) : \underline{\mathrm{CBool}}{\to}\underline{\mathrm{CBool}} \\
\mathrm{and} & \triangleq & \lambda a : \underline{\mathrm{CBool}}.\lambda b : \underline{\mathrm{CBool}}.(a\ \underline{\mathrm{CBool}}\ b\ \underline{\mathrm{ff}}) : \underline{\mathrm{CBool}}{\to}\underline{\mathrm{CBool}}{\to}\underline{\mathrm{CBool}} \\
\mathrm{or} & \triangleq & \lambda a : \underline{\mathrm{CBool}}.\lambda b : \underline{\mathrm{CBool}}.(a\ \underline{\mathrm{CBool}}\ \underline{\mathrm{tt}}\ b) : \underline{\mathrm{CBool}}{\to}\underline{\mathrm{CBool}}{\to}\underline{\mathrm{CBool}}
\end{array}$$

Definizione 5.1 (Tipo induttivo)**.** *Dati i tipi:*

$$U_k \triangleq T_1{\to}T_2{\to}\cdots{\to}T_{n_k}{\to}X \qquad 1 \le k \le p$$

dei costruttori per X, il tipo induttivo *corrispondente è il tipo:*

$$\forall X.U_1{\to}U_2{\to}\cdots{\to}U_p{\to}X$$

Un altro esempio è dato dalla codifica induttiva dei naturali: i costruttori di cui abbiamo bisogno sono rispettivamente quello del successore e quello dello zero.

$$\begin{array}{rcl}
\mathrm{Nat} & \triangleq & \forall X.(X{\to}X){\to}X{\to}X \\
\mathrm{n} & \triangleq & \Lambda X.\lambda f : X{\to}X.\lambda x : X.(f^{\underline{n}}\ x) : \underline{\mathrm{Nat}} \\
\mathrm{succ} & \triangleq & \lambda n : \underline{\mathrm{Nat}}.\Lambda X.\lambda f : X{\to}X.\lambda x : X.(f(n\ X\ f\ x)) : \underline{\mathrm{Nat}}{\to}\underline{\mathrm{Nat}} \\
\mathrm{add} & \triangleq & \lambda n : \underline{\mathrm{Nat}}.\lambda m : \underline{\mathrm{Nat}}.(n\ \underline{\mathrm{Nat}}\ \underline{\mathrm{succ}}\ m) : \underline{\mathrm{Nat}}{\to}\underline{\mathrm{Nat}}{\to}\underline{\mathrm{Nat}}
\end{array}$$

Con il solo ausilio queste codifiche, note anche come *codifiche di Berarducci e Böhm*, è possibile esprimere una vasta gamma di operatori e tipi, come il singoletto, le liste, le coppie, le somme, la ricorsione primitiva, ...

5.1 Proprietà

Teorema 5.2 (Subject reduction)**.**

$$Se \quad \Gamma \vdash t : T \quad e \quad t \to t' \quad allora \quad \Gamma \vdash t' : T$$

Dimostrazione. Procede come al solito, per induzione sul giudizio di tipo $\Gamma \vdash t : T$ e per casi su $t \to t'$. Fondamentali sono i due lemmi seguenti. □

Lemma 5.3 (Sostituzione)**.**

$$Se \quad \Gamma, x : S \vdash t : T \quad e \quad \Gamma \vdash s : S \quad allora \quad \Gamma \vdash t[s/x] : T$$

Lemma 5.4 (Sostituzione al II ordine)**.** *Sia X una variabile di tipo:*

$$se \quad \Gamma \vdash t : T \quad allora \quad \Gamma[S/X] \vdash t[S/X] : T[S/X]$$

e le derivazioni dei giudizi di tipo hanno la stessa altezza.

Dimostrazione. Per induzione sull'altezza di $\Gamma \vdash t : T$. Tutti i casi del prim'ordine sono un'applicazione diretta dell'ipotesi induttiva. I casi rimanenti sono:

- Applicazione II. Sia $Y \not\equiv X$ e $Y \notin FV(S)$

$$\frac{\Gamma \vdash t_1 : \forall Y.T'}{\Gamma \vdash t_1\ R : T'[R/Y]}$$

  Per ipotesi induttiva, abbiamo $\Gamma[S/X] \vdash t_1[S/X] : (\forall Y.T')[S/X] \equiv \forall Y.T'[S/X]$. La regola di applicazione diventa:

$$\frac{\Gamma[S/X] \vdash t_1[S/X] : \forall Y.T'[S/X]}{\Gamma[S/X] \vdash t_1[S/X]\ R[S/X] : T'[S/X][\frac{R[S/X]}{Y}]}$$

  Ora è facile mostrare che:

  - $(t_1[S/X]\ R[S/X]) \equiv (t_1\ R)[S/X] \equiv t[S/X]$;
  - $T'[S/X][\frac{R[S/X]}{Y}] \equiv T'[\frac{R[S/X]}{Y}][S/X] \equiv T'[R/Y][S/X] \equiv T[S/X]$.

- Astrazione II.

$$\frac{\Gamma \vdash t_1 : T_1 \qquad Y \notin FV(\Gamma)}{\Gamma \vdash \Lambda Y.t_1 : \forall Y.T_1}$$

  Sia Z fresca in Γ e S. Per ipotesi induttiva:

$$\Gamma[Z/Y] \vdash t_1[Z/Y] : T_1[Z/Y]$$

con la stessa altezza di $\Gamma \vdash t_1 : T_1$. Inoltre, siccome $Y \notin FV(\Gamma)$, abbiamo $\Gamma[Z/Y] \equiv \Gamma$. Quindi, per ipotesi induttiva e riapplicando la regola di astrazione:

$$\frac{\Gamma[S/X] \vdash t_1[Z/Y][S/X] : T_1[Z/Y][S/X] \qquad Z \notin FV(\Gamma)}{\Gamma[S/X] \vdash \Lambda Z.t_1[Z/Y][S/X] : \forall Z.T_1[Z/Y][S/X]}$$

Ora, come prima, abbiamo:

- $\Lambda Z.t_1[Z/Y][S/X] \equiv (\Lambda Z.t_1[Z/Y])[S/X] \equiv (\Lambda Y.t_1)[S/X] \equiv t[S/X]$;
- $\forall Z.T_1[Z/Y][S/X] \equiv (\forall Z.T_1[Z/Y])[S/X] \equiv (\forall Y.T_1)[S/X] \equiv T[S/X]$.

□

Teorema 5.5 (Forte normalizzazione)**.** *Nel sistema F, vale che, per la full-β-reduction, ogni sequenza di riduzioni è finita. In weak call-by-value, la riduzione di un termine ben tipizzato è anch'essa finita. La dimostrazione di questo teorema segue il metodo dei "candidati di riducibilità" (vedi Girard et al. [1989]).*

Tabella riassuntiva della proprietà di forte normalizzazione per alcune varianti di λ-calcoli tipizzati:

Calculus	*s.n.*
Simply typed λ-calculus	$\surd$
System T	$\surd$
System F o λ_2	$\surd$
λ_3	$\surd$
λ_4	$\surd$
...	
System $F\omega$	$\surd$

Per una classificazione ed uno studio formale delle principali varianti di λ-calcoli con tipi, vedi Barendregt et al. [1992].

Teorema 5.6 (Espressività)**.** *Le funzioni definibili da λ-termini del II ordine, con tipo $\underline{Nat} \to \underline{Nat}$ dove:*

$$Nat \triangleq \forall X.(X \to X) \to X \to X$$

sono tutte e sole le funzioni da naturali in naturali dimostrabilmente totali nell'aritmetica del second'ordine:

$$PA_2 \vdash \forall n.\exists m.f(n){=}m$$

5.2 Quantificatore esistenziale

Come per il tipo quantificatore universale, possiamo inserire il tipo esistenziale.

`[ M ]` $t ::= \ldots \mid \texttt{let } (X,x)\texttt{=}t \texttt{ in } t \mid (T,t)$

`[ E ]`

$$\texttt{let } (X,x)\texttt{=}(S,t) \texttt{ in } s \to s[S/X][t/x] \quad (\textsc{Eunpack})$$

$$\frac{t \to t'}{(S,t) \to (S,t')}\ (\textsc{Epack}^{\text{ctx}})$$

$$\frac{t \to t'}{\texttt{let } (X,x)\texttt{=}t \texttt{ in } s \to \texttt{let } (X,x)\texttt{=}t' \texttt{ in } s}\ (\textsc{Eunpack}^{\text{ctx}})$$

La chiusura su s non è necessaria, perchè siamo in call-by-value.

`[ S ]` $T ::= \ldots \mid \exists X.T$

`[ T ]` Inseriamo due nuove regole di tipizzazione:

$$\frac{\Gamma \vdash t : T[S/X]}{\Gamma \vdash (S,t) : \exists X.T}\ (\text{T}\exists^{\text{intro}})$$

$$\frac{\Gamma \vdash t : \exists X.T \qquad \Gamma, x : T \vdash s : S \qquad X \notin FV(\Gamma, S)}{\Gamma \vdash \texttt{let } (X,x)\texttt{=}t \texttt{ in } s : S}\ (\text{T}\exists^{\text{elim}})$$

In questo calcolo, i tipi esistenziali corrispondono alle "interfacce", cioè ai tipi "astratti", mentre le coppie (Tipo, Termine) sono le implementazioni. Ad esempio, il tipo "astratto", esistenziale $\exists X.X \to X$ è abitato da `(Nat,succ)`, ma anche da `(Bool,not)`. Il termine:

$$\texttt{let } (X,x)\texttt{=}(\text{Nat},\text{succ}) \texttt{ in } (x\ \underline{3})$$

rappresenta un uso erratto del tipo astratto, perchè dipende dall'istanza che se ne farà (infatti sarebbe stato errato se l'avessimo istanziato con `(Bool,not)`, perchè non si può applicare un operatore booleano all'intero $\underline{3}$).

5.3 Ricostruzione del tipo

Si dimostra che il problema di ricostruzione del tipo è indecidibile nel sistema F, anche nelle forme più deboli, ad esempio decidere che tipo bisogna mettere nelle applicazioni di tipo. Per una trattazione completa: Wells [1996]; Urzyczyn [1997].

Appendice A

Correttezza e completezza dell'algoritmo di subtyping per tipi ricorsivi (M. Patrignani)

Teorema A.1 (Correttezza)**.** *Dati due μ-tipi S, T, se l'esecuzione di sub$(\emptyset, S, T)$ termina, allora $S <: T$, cioè la coppia (S, T) appartiene al massimo punto fisso dell'operatore S_t:*

$$Se \quad sub(\emptyset, S, T) = \Gamma' \quad allora \quad (S, T) \in \nu S_t$$

Dimostrazione. Cominciamo analizzando il ramo a destra dell'implicazione:

$$(S, T) \in \nu S_t$$

è dunque sufficiente dimostrare che esiste un insieme $\Delta \subseteq \mathscr{T}_m \times \mathscr{T}_m$ tale che:

$$(S, T) \in \Delta \quad \text{e} \quad \Delta \subseteq \nu S_t.$$

cioè esiste un insieme Δ di coppie di μ-tipi tali che la coppia (S, T) appartiene a Δ e inoltre $\Delta \subseteq S_t(\Delta)$. Applicando il principio di coinduzione otteniamo:

$$\frac{\Delta \subseteq S_t(\Delta)}{\Delta \subseteq \nu S_t}$$

La scelta più ovvia è prendere $\Delta = \Gamma'$ visto che Γ' contiene (S, T) come si può facilmente dedurre dall'algoritmo stesso: infatti prima di ritornare, *sub* aggiunge sempre la coppia (S, T) all'insieme d'ipotesi in output. Il nostro enunciato diventa dunque:

$$\text{se} \quad \text{sub}(\emptyset, S, T) = \Gamma' \quad \text{allora} \quad \Gamma' \subseteq S_t(\Gamma').$$

Procediamo per induzione sull'esecuzione di *sub*, notando che se termina è perchè compie un numero finito di chiamate ricorsive, quindi la quantità che osserviamo e che va calando è il numero di chiamate ricorsive da fare. L'ipotesi induttiva fornitaci a questo punto però è troppo debole per dimostrare i passi induttivi, pertanto qui si dimostra un fatto più forte che implica il teorema di correttezza:

$$\forall R \begin{cases} sub(\Gamma, S, T) = \Gamma' \\ \Gamma \subseteq S_t(\Gamma \cup \{(S,T)\} \cup R) \end{cases} \implies \Gamma' \subseteq S_t(\Gamma' \cup R)$$

Come si può facilmente notare infatti, l'enunciato base di correttezza corrisponde al caso R$\equiv \emptyset$.
Dimostriamo ora questo fatto per induzione sul numero di chiamate ricorsive nell'esecuzione di *sub*.

Casi base: Vi sono due casi in cui l'algoritmo termina dopo una sola chiamata:

$(S,T) \in \Gamma$ **:**
in questo caso *sub* termina restituendo Γ, quindi abbiamo che

$$\Gamma' \equiv \Gamma$$

pertanto la conclusione deriva direttamente dalle ipotesi:

$$\forall R \begin{cases} sub(\Gamma, S, T) = \Gamma \\ \Gamma \subseteq S_t(\Gamma \cup R) \end{cases} \implies \Gamma \subseteq S_t(\Gamma \cup R)$$

$T \equiv \texttt{Top}$ **:**
in questo caso invece *sub* ritorna Γ_0 dove

$$\Gamma_0 \equiv \Gamma \cup \{(S, \texttt{Top})\}$$

quindi bisogna dimostrare che:

$$\forall R \begin{cases} sub(\Gamma, S, \texttt{Top}) = \Gamma_0 \\ \Gamma \subseteq S_t(\Gamma \cup \{(S, \texttt{Top})\} \cup R) \end{cases} \implies \Gamma_0 \subseteq S_t(\Gamma_0 \cup R)$$

La tesi, sostituendo Γ_0 con $\Gamma \cup (S, \texttt{Top})$, diventa:

$$\Gamma \cup (S, \texttt{Top}) \subseteq S_t(\Gamma \cup (S, \texttt{Top}) \cup R)$$

ora Γ è ovviamente contenuto per ipotesi nel membro destro, mentre la coppia $(S, \texttt{Top})$ vi appartiene per la prima regola dell'operatore S_t, che aggiunge tutte le coppie $(T, \texttt{Top})$ purchè T sia in $\mathscr{T}_m$.

Passo induttivo: Un passo induttivo consta di quattro casi che prevedono la considerazione delle chiamate ricorsive che abbiamo in presenza di $\times$,$\rightarrow$, unfolding su T e su S.

$S \equiv S_1 \times S_2 \wedge T \equiv T_1 \times T_2$

La chiamata a *sub* in questo caso si risolve in quattro passi algoritmici:

1. $\Gamma_0 = \Gamma \cup (S_1 \times S_2, T_1 \times T_2)$;
2. $\Gamma 1 = sub(\Gamma_0, S_1, T_1)$;
3. $\Gamma 2 = sub(\Gamma 1, S_2, T_2)$;
4. $return\ \Gamma 2$;

precediamo applicando l'ipotesi induttiva al passo 2.

$$\forall R' \begin{cases} sub(\Gamma_0, S_1, T_1) = \Gamma 1 \\ \Gamma_0 \subseteq S_t(\Gamma_0 \cup (S_1, T_1) \cup R') \end{cases} \implies \Gamma 1 \subseteq S_t(\Gamma 1 \cup R')$$

La prima condizione è soddisfatta per le garanzie di terminazione che abbiamo, per la seconda occorre effettuare la sostituzione sintattica di Γ_0 col termine che si ottiene al passo algoritmico 1.

$$\Gamma \cup (S_1 \times S_2, T_1 \times T_2) \subseteq S_t(\Gamma \cup (S_1 \times S_2, T_1 \times T_2) \cup (S_1, T_1) \cup R')$$

Siccome siamo sotto l'assunzione che valga per ogni R', scegliamo $R' \equiv \mathrm{R} \cup (S_2, T_2)$, cosi da poter rendere vera la formula.
Infatti l'unico problema sarebbe che negli argomenti passati a S_t non vi sono le ipotesi per creare la coppia $(S_1 \times S_2, T_1 \times T_2)$, tuttavia, aggiungendo la coppia (S_2, T_2), e possiamo farlo in quanto stiamo utilizzando un generico R', si creano le ipotesi per applicare la regola 2 dell'operatore e ottenere la coppia desiderata.
A questo punto, applicando l'ipotesi induttiva, otteniamo che

$$\Gamma 1 \subseteq S_t(\Gamma 1 \cup (S_2, T_2) \cup R)$$

Applichiamo l'ipotesi induttiva ora al passo algoritmico 3:

$$\forall R'' \begin{cases} sub(\Gamma 1, S_2, T_2) = \Gamma 2 \\ \Gamma 1 \subseteq S_t(\Gamma 1 \cup (S_2, T_2) \cup R'') \end{cases}$$

Ancora, il primo asserto è vero, dobbiamo quindi verificare il secondo. Questo ci è garantito esserlo in quanto coincide perfettamente con ciò che abbiamo appena ottenuto dall'applicazione dell'ipotesi induttiva al passo algoritmico 2, prendendo $R'' \equiv R$.

Ancora una volta possiamo fare questa sostituzione in quanto stiamo supponendo che gli asserti trattati valgano per ogni R. Concludiamo quindi che:

$$\Gamma 2 \subseteq S_t(\Gamma 2 \cup R'')$$

Questo basta a concludere la dimostrazione del passo induttivo in quanto $\Gamma 2$ è l'output, oltre che del passo di lunghezza n, anche di quello di lunghezza *n+1*, quindi dall'applicazione dell'ipotesi induttiva al passo algoritmico 3, otteniamo la soddisfazione della tesi generale:

$$\forall R \begin{cases} sub(\Gamma, S_1 \times S_2, T_1 \times T_2) = \Gamma' \\ \Gamma \subseteq S_t(\Gamma \cup (S_1 \times S_2, T_1 \times T_2) \cup R) \end{cases} \implies \Gamma' \subseteq S_t(\Gamma' \cup R)$$

dove $\Gamma' \equiv \Gamma 2$ e R $\equiv R''$.

$S \equiv S_1 \to S_2 \wedge T \equiv T_1 \to T_2$
la dimostrazione per questo caso è analoga alla precedente tranne per il fatto che bisogna applicare controvarianza alla prima chiamata ricorsiva.

$T \equiv \mu X.T_1$
In questo passo dobbiamo dimostrare che:

$$\forall R \begin{cases} sub(\Gamma, S, \mu X.T_1) = \Gamma' \\ \Gamma \subseteq S_t(\Gamma \cup (S, \mu X.T_1) \cup R) \end{cases} \implies \Gamma' \subseteq S_t(\Gamma' \cup R)$$

I passi algoritmici da analizzare sono solo 2:

1. $\Gamma_0 = \Gamma \cup (S, \mu X.T_1)$;
2. $sub(\Gamma_0, S, T_1[T/X])$;

Come si può notare, l'output della chiamata al passo *n+1* coincide con quello del passo n. Dimostriamo di poter applicare l'ipotesi induttiva a quel passo per poter concludere la validità del teorema. L'ipotesi induttiva è la seguente:

$$\forall R' \begin{cases} sub(\Gamma_0, S, T_1[T/X]) = \Gamma' \\ \Gamma_0 \subseteq S_t(\Gamma_0 \cup (S, T_1[T/X]) \cup R') \end{cases} \implies \Gamma' \subseteq S_t(\Gamma' \cup R')$$

La terminazione e quindi la validità del primo asserto ci è garantita per ipotesi di terminazione generale. Effettuiamo la sostituzione sintattica di Γ_0 per dimostrare che anche l'altra ipotesi è soddisfatta e l'ipotesi induttiva è applicabile.

$$\Gamma \cup (S, \mu X.T_1) \subseteq S_t(\Gamma \cup (S, \mu X.T_1) \cup (S, T_1[T/X]) \cup R)$$

L'unico problema potrebbe essere dato dalla generazione della coppia $(S, \mu X.T_1)$ ma vediamo che è esattamente ciò che otteniamo dalla quarta regola dell'operatore. Vediamo poi che le ipotesi per applicare questa regola sono soddisfatte quindi l'ipotesi induttiva è applicabile.
A questo punto otteniamo che:

$$\Gamma' \subseteq S_t(\Gamma' \cup R)$$

che è esattamente la conclusione del teorema.

$S \equiv \mu X.S_1$
la dimostrazione per questo caso è analoga alla precedente.

□

Teorema A.2 (Completezza)**.** *Se l'esecuzione di* $sub(\emptyset, S, T)$ *ritorna* `fail`*, allora la coppia* (S, T) *non appartiene al massimo punto fisso di* S_t*.*

$$sub(\emptyset, S, T) = \texttt{fail} \implies (S, T) \notin \nu S_t$$

Dimostrazione. Anche in questo caso ci riconduciamo a dimostrare qualcosa di più forte che implichi l'enunciato del teorema:

$$\forall \Gamma : \mathcal{P}(\mathscr{T}_m \times \mathscr{T}_m)$$

$$\text{sub}(\Gamma, S, T) = \texttt{fail} \implies (S, T) \notin \nu(S_t)$$

La dimostrazione procede per assurdo.
Supponiamo che l'algoritmo possa fallire con una coppia che appartiene al massimo punto fisso di S_t. Supponiamo quindi $(S, T) \in \nu(S_t)$.
Esplicitando il ramo destro dell'implicazione otteniamo che:
$\exists R : \mathcal{P}(\mathscr{T}_m \times \mathscr{T}_m)$ *t.c.*:
$(S, T) \in R$
$R \subseteq \nu(S_t)$ che per coinduzione diventa $R \subseteq S_t(R)$.
Analizziamo i possibili casi nei quali l'algoritmo ritorna `fail`:

1. Al primo passo;

2. Dopo una qualsiasi chiamata ricorsiva.

Nel primo caso quindi la coppia (S, T) non appartiene a Γ e non ha la struttura sintattica di nessuna delle guardie degli *if*, pertanto l'algoritmo ritorna `fail`. Dalle ipotesi però sappiamo che $(S, T) \in S_t(R)$, quindi, per costruzione ha la precisa struttura sintattica di una delle cinque guardie degli *if*.

Ecco quindi l'assurdo per il primo caso.
Nel secondo caso consideriamo un'ipotetica chiamata ricorsiva terminante in `fail`. Tutte le chiamate ricorsive sono fatte su elementi(i vari (S_1, T_1), (S_2, T_2), $(S, T1[T/X])$, ecc...) che appartengono ad R.
Esemplifichiamo: se $S \equiv S_1 \times S_2 \wedge T \equiv T_1 \times T_2$, e $(S, T) \in R$, allora per definizione di S_t, anche $(S_1, T_1), (S_2, T_2) \in R$.
A questo punto si può applicare il ragionamento fatto al punto precedente, se la struttura sintattica delle sotto coppie non è coperta da una guardia degli *if* e $(S_1, T_1), (S_2, T_2) \notin \Gamma$, allora non è possibile che appartengano al massimo punto fisso di S_t e si giunge all'assurdo.
Avendo dimostrato l'assudo per ogni possibile caso, è chiaro che l'assunzione che una coppia possa appartenere al massimo punto fisso di S_t e che *sub* restituisca `fail` con essa è errato.

L'unica nota da chiarire a questo punto è sull'effettiva terminazione dell'algoritmo. L'agoritmo potrebbe divergere solo a causa degli unfolding, questo però non accade grazie al fatto che lavoriamo con μ-tipi contrattivi. □

Bibliografia

Barendregt, H., Abramsky, S., Gabbay, D. M., Maibaum, T. S. E., and Barendregt, H. P. 1992. Lambda calculi with types. In *Handbook of Logic in Computer Science*, pages 117–309. Oxford University Press.

Barendregt, H. P. 1981. *The lambda calculus: its syntax and semantics / H.P. Barendregt*. North-Holland Pub. Co.; sole distributors for the U.S.A. and Canada Elsevier North-Holland, Amsterdam; New York.

Church, A. 1940. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5(2):56–68.

Curry, H. B. 1958. *Combinatory logic / [by] Haskell B. Curry [and] Robert Feys. With two sections by William Craig*. North-Holland Pub. Co., Amsterdam.

Gallier, J. 1993. Constructive logics part I: A tutorial on proof systems and typed–calculi. *Theoretical Computer Science*, 110:249–339.

Girard, J.-Y., Taylor, P., and Lafont, Y. 1989. *Proofs and types*. Cambridge University Press, New York, NY, USA.

Harper, R. 2005. Programming in standard ML.

Hindley, J. R. and Seldin, J. P. 1986. *Introduction to Combinators and λ-calculus*, volume 1 of *London Mathematical Society – Student Texts*. Cambridge University Press.

Howard, W. A. 1980. The formulae-as-types notion of construction. In Seldin, J. P. and Hindley, J. R., editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press.

Milner, R., Tofte, M., Harper, R., and Macqueen, D. 1997. *The Definition of Standard ML - Revised*. The MIT Press.

Pierce, B. C. 2002. *Types and Programming Languages*. MIT Press.

Sørensen, M. H. and Urzyczyn, P. 2006. *Lectures on the Curry-Howard Isomorphism, Volume 149 (Studies in Logic and the Foundations of Mathematics)*. Elsevier Science Inc., New York, NY, USA.

Urzyczyn, P. 1997. Type reconstruction in Fω. *Mathematical Structures in Computer Science (MSCS)*, 7(4):329–358.

Wells, J. B. 1996. Typability and type checking in the second-order lambda-calculus are equivalent and undecidable. In *In Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 176–185. Society Press.