Appunti di Informatica Teorica
(Prof. Ricci a.a. 2005–2006)

Fabio Biselli

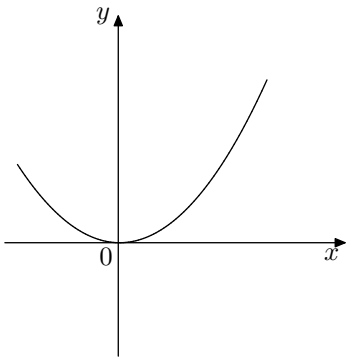
Indice

Capitolo 1

Introduzione al calcolo dei combinatori

1.1 Funzioni: due punti di vista

1. *visione estensionale:* insieme di coppie $\langle x, y \rangle$



2. *visione operativa:* la funzione $y = x^2 - x$ viene vista come un programma: prendi x e calcolane il quadrato, sottrai quindi x ed ottieni y.

Nelle funzioni aritmetiche e nell'analisi le due visioni si equivalgono, ma vedremo che non è sempre così.

Esempio. 1 *La funzione più semplice è la funzione identità* $\mathbf{I}(x) = x$. *Nella visione operativa questa, dato un argomento* x, *restituisce l'argomento stesso. La visione estensionale invece formalizza l'insieme* $\{ \langle x, x \rangle \mid x \in X \text{ per qualche } X \}$.

1.1.1 Visione estensionale

Nella Teoria degli Insiemi (secondo il testo consigliato, J. D. Monk, Introduzione alla teoria degli insiemi, Boringhieri, 1972.) i postulati formalizzano la nozione di appartenenza, indicata dal simbolo '$\in$' ($x \in X$); da essa derivano quella di insieme ("insieme piccolo") e di classe propria ("insieme grande"). Gli insiemi piccoli sono quegli oggetti x che stanno a sinistra del simbolo di appartenenza cioè gli x tale che x appartiene a X per qualche X. Le classi sono quegli oggetti che stanno a destra cioè gli X per cui x appartiene a X per qualche x. Se una classe non è un insieme si dice propria. Un caso di tali classi proprie è quello del nostro $\mathbf{I}$.

Per le funzioni esiste la nozione di applicazione di un argomento x (che può anche essere una funzione) a una funzione, ottenendo un insieme $f(x)$. Quando x non appartiene all'insieme degli argomenti effettivi di f, cioè al dominio $\mathrm{Dom}\, f = \{\, x \mid \langle x, y\rangle \in f \text{ per qualche } y \,\}$, il valore di $f(x)$ è un ? che insiemisticamente è espresso nella classe propria totale V:

$$V = \{\, x \mid x \text{ è un insieme} \,\}.$$

Pertanto $\mathbf{I}(\mathbf{I}) = V$ poiché $\mathbf{I}$ non appartiene al $\mathrm{Dom}\, \mathbf{I}$ (altrimenti non sarebbe classe propria). Si noti che $\mathbf{I} \neq V$.

Consideriamo ora l' idea della funzione identità dal punto di vista operativo: lascia le cose come sono. Tale *funzione operativa* $\mathbf{I}$ avrà la proprietà che:

$$\mathbf{I}(\mathbf{I}) = \mathbf{I}.$$

Già nella funzione più semplice le due visioni sono differenti.

1.1.2 Visione operativa

Esistono solo regole operative (non oggetti come insiemi etc.). Tali regole sono formalizzate da come operano fra loro. Ciò avviene in due modi distinti: calcolo dei combinatori e λ-calcolo.

Nel calcolo dei combinatori si assumono delle funzioni operative elementari con cui si costruiscono tutte le altre tramite una operazione detta di applicazione e si introducono delle nozioni di uguaglianza tra tali funzioni operative dipendenti da nozioni di calcolo effettivo dette *riduzione*. Nel λ-calcolo non si assumono delle funzioni elementari, nel senso che esse siano da sole operativamente definite, ma solo degli oggetti su cui si opera con una "applicazione" e un'astrazione che sarà in un certo senso l'opposto dell'applicazione.

Nel primo modo si costruiscono così degli oggetti, termini combinatori, che corrispondono ai programmi di un linguaggio di programmazione (imperativo) idealizzato che operi con delle funzioni predefinite (elementari) mediante applicazione di"dati" a una procedura. Tali dati possono essere procedure. Non sono ammessi richiami di sottoprocedure con parametri.

Nel λ-calcolo i λ-termini invece non hanno funzioni predefinite da cui partire, avranno invece le "call" (procedure con parametri richiamabili).

1.2 Calcolo dei combinatori

Si assume l'esistenza di una sequenza di cosiddette costanti che contenga almeno tre elementi: **I**, **K**, **S**. Si assume l'esistenza di una sequenza numerabile infinita di cosiddette variabili: x, y, z, u, v, etc. Per atomo si intenderà sia una costante che una variabile, i termini combinatori sono definiti dalle seguenti clausole che usano un'operazione di applicazione fra due termini X e Y che definiremo dopo e che indicheremo ora con l'accostamento XY:

a) ogni atomo è un termine combinatorio;

b) se X, Y sono due termini combinatori, allora anche l'applicazione XY è un termine combinatorio.

L'applicazione è definita dai seguenti postulati, ove per termine composto si intende un termine che sia l'applicazione di altri due termini combinatori:

1) Le costanti, le variabili ed i termini composti formano tre "insiemi" disgiunti.

2) Se due termini composti sono le stesso termine, allora così è ordinatamente per i termini componenti:

$$X'Y' \equiv X''Y'' \Longrightarrow X' \equiv X'' \text{ e } Y' \equiv Y'' \tag{1.1}$$

Si intende inoltre che nessun altro oggetto è un termine combinatorio se non proviene dalle clausule a) e b).

Stiamo usando "$\equiv$" perchè riserviamo "$=$" a delle equivalenze che, pur non essendo l'identità, saranno più frequenti. Stiamo usando la parola insieme nel senso insiemistico, ma potremmo farne a meno. Insiemisticamente potremmo creare un modello dei termini combinatori prendendo come applicazione l'accoppiamento e come atomi opportuni insiemi. (L'accoppiamento insiemistico gode di maggiori proprietà di quelle dell'applicazione, ma queste non ci servono).

Sempre (ab)usando nozioni insiemistiche potremmo, indicando con T l'insieme dei termini composti, concludere che:

- ogni atomo è un termine combinatorio;

- se $X'Y' = X''Y''$ allora $X' = X''$ e $Y' = Y''$;

- per ogni T' contenuto in T vale che $T' = T$ se e solo se ogni atomo appartiene a T' e se X', Y' appartengono a T' allora $X'Y'$ appartiene a T.

Tale lista di proprietà corrisponde a quella dei postulati di Peano:

- 0 è un numero naturale;

- se $n + 1 = m + 1$ allora $n = m$;

- per ogni N' contenuto in N vale che $N' = N$ sse ($0 \in N'$, $n \in N' \Rightarrow n + 1 \in N'$).

Le due liste di proprietà differiscono "di poco":

- i numeri naturali hanno un solo zero, mentre i termini combinatori ne possono avere molti (gli atomi).
- i numeri naturali provengono dallo zero mediante la funzione successore che è una operazione unaria (ad un argomento), mentre per i termini combinatori si usa l'applicazione che è binaria (a due argomenti).

Per il resto nulla cambia.

Capitolo 2

Calcolo dei combinatori

Dall'analogia relativa ai postulati di Peano tra Termini Combinatori e numeri naturali possiamo dire che nella maggior parte dei casi per definire una funzione o un predicato, relativi ai termini combinatori, procederemo con una base di definizione (le funzioni o predicati vengono definiti per tutti gli atomi) e un passo induttivo (se la funzione o il predicato si suppongono definiti per due termini combinatori si da la costruzione per definirli sull'applicazione dei due termini).

Lo stesso varrà per le dimostrazioni. Nella base della dimostrazione si dimostra l'asserto per gli atomi, nel passo induttivo, presunta la verità dell'asserto per due termini, si dedurrà l'asserto per la loro applicazione.

Per i termini combinatori conviene, sopratutto all'inizio, considerarne la rappresentazione grafica mediante alberi binari con foglie etichettate dagli atomi.

Esempio. 2 *Alcune rappresentazioni grafiche.*

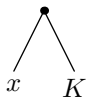


Figura 2.1: Rappresentazione grafica di $x\mathbf{K}$.

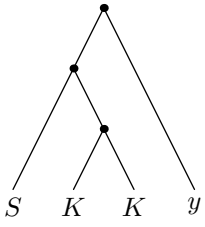


Figura 2.2: Rappresentazione grafica di $(\mathbf{S}(\mathbf{KK}))y$.

Nella definizione di termini combinatori non compaiono parentesi. Le parentesi che scriviamo sono solo sintattiche (servono ad indicare le precedenze nella costruzione). Invece nella notazione dell'Analisi (espressa nella Teoria degli Insiemi) le parentesi possono servire ad indicare l'applicazione (insiemistica) di un argomento ad una funzione: $f(x)$, $[M_v]_s$, etc. Ciò fa sì che nella nostra notazione ci siano meno parentesi di quelle dell'Analisi. Tuttavia restano nei termini combinatori più grossi un buon numero di parentesi.

Si può allora adottare una convenzione di semplificazione di scrittura che le riduce considerevolmente pur evitando ambiguità: *in ogni termine o sottotermine si omettono le coppie di parentesi associate a sinistra.*

Esempio. 3 $(\mathbf{S}(\mathbf{KK}))y$ *diventa* $\mathbf{S}(\mathbf{KK})y$.

Data una scrittura semplificata si può sempre risalire univocamente alla scrittura non semplificata o meglio all'albero.

Esempio. 4 *Ecco graficamente come vegono rappresentati i seguenti termini:*

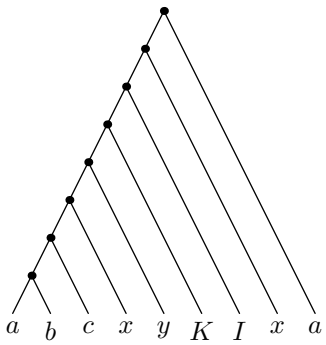


Figura 2.3: $abcxy\mathbf{KI}xa \longrightarrow (((((((ab)c)x)y)\mathbf{K})\mathbf{I})x)a$.

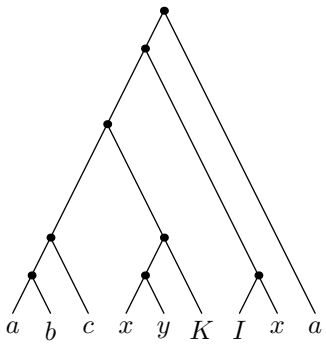


Figura 2.4: $abc(xy\mathbf{K})(\mathbf{I}x)a \longrightarrow (((ab)c)(((xy)\mathbf{K}))(\mathbf{I}x))a$.

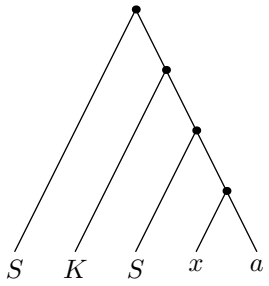


Figura 2.5: $\mathbf{S}(\mathbf{K}(\mathbf{S}(xa)))$ resta invariata.

2.1 Operazioni su termini combinatori

In questa parte del Capitolo parleremo delle operazioni possibili sui termini combinatori, in particolare della sostituzione, sostituzioni multiple, riduzione, astrazione e riduzione multipla.

2.1.1 Sostituzione

Def. 2.1 *Per i termini combinatori si può definire un'operazione di* sostituzione *che da una variabile e due termini combinatori restituisce un altro termine combinatorio. Siano N ed M termini combinatori ed x una variabile,* la sostituzione di N al posto di x in M *è il termine combinatorio $[N/x]M$ definito da:*

1. $[N/x]a \equiv a$ *se* a *è un atomo* $\neq x$;
2. $[N/x]x \equiv N$;
3. $[N/x]UV \equiv ([N/x]U)([N/x]V)$.

Talvolta base e passo induttivo sembrano mescolati.

Def. 2.2 *Diciamo che un termine combinatorio U* occorre *in un termine combinatorio X e scriviamo $U \in X$ (non è un'appartenenza) qualora ciò può essere dedotto da:*

a) $U \in U$;

b) Se $U \in X$ *o se* $U \in Y$ *allora* $U \in XY$.

Anche questa è una definizione induttiva del predicato $U \in X$ con induzione su U. Infatti in a) si dà (anche) la base (U può essere un'atomo) mentre il passo induttivo viene dato sia da b) che da a).

Esempio. 5 *Vediamo esempi di sostituzioni e di* $\in$.

Sia $M \equiv S(KK)(xI)$ *e calcoliamo la sostituzione di* $[yS/x]M$

$$\begin{array}{rcl}
[yS/x]M & \overset{3}{\equiv} & ([yS/x]S(KK))([yS/x]xI) \\
& \overset{3}{\equiv} & (([yS/x]S)([yS/x]KK))([yS/x]xI) \\
& \overset{1}{\equiv} & (S([yS/x]KK))([yS/x]xI) \\
& \overset{3}{\equiv} & (S(([yS/x]K)([yS/x]K)))([yS/x]xI) \\
& \overset{1}{\equiv} & (S(KK)([yS/x]xI) \\
& \overset{3}{\equiv} & (S(KK)(([yS/x]x)([yS/x]I))) \\
& \overset{2}{\equiv} & (S(KK)((yS)([yS/x]I))) \\
& \overset{1}{\equiv} & (S(KK)((yS)I)) \\
& \equiv & S(KK)(ySI).
\end{array}$$

Lo si sarebbe potuto individuare subito utilizzando la rappresentazione grafica:

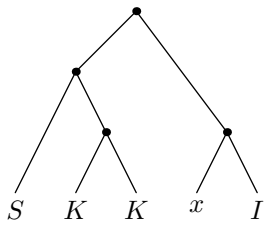


Figura 2.6: Rappresentazione grafica di M.

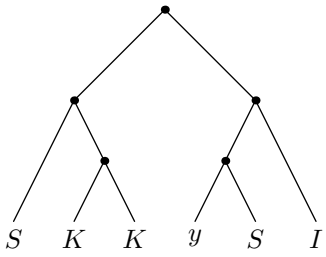


Figura 2.7: Rappresentazione grafica di $[y\mathbf{S}/x]M$.

Oss. 2.1 *La definizione di sostituzione riguarda solo atomi* x *che siano variabili (è questa la ragione della distinzione fra variabili e costanti). In particolare non si possono sostituire termini al posto delle costanti riducibili. Come vedremo in seguito anche una tale "sostituzione" può avere significato.*

Esercizio. 1 *Verifichiamo se* $\mathbf{KK} \in \mathbf{S}(\mathbf{KK})(x\mathbf{I})$ *e se* $\mathbf{KK}(x\mathbf{I}) \overset{?}{\in} \mathbf{S}(\mathbf{KK})(x\mathbf{I})$.

1) $\mathbf{KK} \in \mathbf{S}(\mathbf{KK})(x\mathbf{I})$.

per b) $\mathbf{KK} \stackrel{?}{\in} \mathbf{S}(\mathbf{KK})$ *oppure* $\mathbf{KK} \stackrel{?}{\in} x\mathbf{I}$.

Per quanto riguarda l'occorrenza di destra potremmo andare avanti come prima, ma non potremmo concludere che vi sia. Andiamo quindi ad analizzare l'occorrenza di sinistra:

per b) $\mathbf{KK} \stackrel{?}{\in} \mathbf{S}$ *oppure* $\mathbf{KK} \stackrel{?}{\in} \mathbf{KK}$.

$\mathbf{KK} \notin \mathbf{S}$ *poichè* $\mathbf{S}$ *è atomo e* $\mathbf{KK}$ *è termine composto.*

$\mathbf{KK} \in \mathbf{KK} \longrightarrow$ *si, per la clausola a).*

Questo corrisponde al fatto che $\mathbf{KK}$ *è sottoalbero di* $\mathbf{S}(\mathbf{KK})(x\mathbf{I})$.

2) $\mathbf{KK}(x\mathbf{I}) \stackrel{?}{\in} \mathbf{S}(\mathbf{KK})(x\mathbf{I})$.

Possiamo fare la deduzione come sopra, ma deduciamo subito graficamente che non c'è occorrenza.

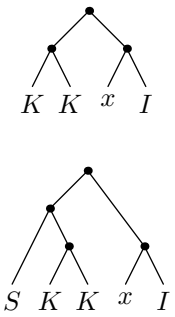


Figura 2.8: $\mathbf{KK}(x\mathbf{I})$ non è sottoalbero di $\mathbf{S}(\mathbf{KK})(x\mathbf{I})$.

Oss. 2.2 *In* $xS(xS)$ *abbiamo due verifiche che* $xS \in xS(xS)$*, per indicare quale dei due sottoalberi ci interessa parleremo di occorrenze di* xS *in* $xS(xS)$.

2.1.2 Sostituzione multipla

In luogo di una variabile si possono contemporaneamente sostituire più variabili. Vediamolo nel caso di due variabili.

Def. 2.3 *Nel caso di due variabili definiamo la sostituzione di un termine combinatorio* N' *al posto della variabile* x *e di un termine combinatorio* N'' *al posto della variabile* y *in un termine* M *ed indicheremo il risultato con* $[N'/x, N''/y]M$ *mediante le seguenti clausole:*

- $[N'/x, N''/y]a \equiv a$ *se* a *è un atomo* $\neq x, y$;
- $[N'/x, N''/y]x \equiv N'$;

- $[N'/x, N''/y]y \equiv N''$;
- $[N'/x, N''/y]UV \equiv ([N'/x, N''/y]U)([N/x, N''/y]V)$.

Doppia induzione.

2.1.3 Riduzione

Def. 2.4 *Diremo che il termine combinario X si* riduce (debolmente) *al termine combinatorio Y e scriveremo $X \rhd Y$, qualora ciò possa essere dedotto dai seguenti assiomi.*

(I) $\mathbf{I}X \rhd X$;

(K) $\mathbf{K}XY \rhd X$;

(S) $\mathbf{S}XYZ \rhd XZ(YZ)$;

(ρ) $X \rhd X$.

Per ogni termine combinatorio X, Y e Z utilizziamo le seguenti regole di deduzione:

- Se $X \rhd Y$ allora $XZ \rhd YZ$;
- se $X \rhd Y$ allora $ZX \rhd ZY$;
- se $X \rhd Y$ *e* $Y \rhd Z$ allora $X \rhd Z$.

Def. 2.5 (redex) *Diremo che un termine di tipo $\mathbf{I}X$, $\mathbf{K}XY$, $\mathbf{S}XYZ$ è un* redex *e che i corrispondenti termini X, X e $XZ(YZ)$ sono i rispettivi ridotti (o contratti) immediati.*

L' uso di uno dei tre assiomi (I), (K), (S) sarà detto contrazione.

Def. 2.6 *Se in un termine combinatorio Y non occorrono dei redex allora diremo che Y è in* forma normale *e se $X \rhd Y$, diremo che Y è una forma normale di X.*

Esempio. 6 *Consideriamo un termine combinatorio $\mathbf{S}(\mathbf{KS})\mathbf{K}$ che chiamiamo $\mathbf{B}$ e con essocostruiamo il termine generico $\mathbf{B}XYZ$ con X, Y, Z termini qualsiasi. Si ha:*

$$\begin{aligned}
\mathbf{B}XYZ &\equiv \underbrace{\mathbf{S}(\mathbf{KS})\mathbf{K}X}_{\text{redex}}YZ \\
&\overset{(S)}{\rhd} \underbrace{\mathbf{KS}X}_{\text{redex}}(\mathbf{K}X)YZ \\
&\overset{(K)}{\rhd} \underbrace{\mathbf{S}(\mathbf{K}X)YZ}_{\text{redex}} \\
&\overset{(S)}{\rhd} \underbrace{\mathbf{K}XZ}_{\text{redex}}(YZ) \\
&\overset{(K)}{\rhd} X(YZ).
\end{aligned}$$

L'esempio precedente mostra una caratteristica desiderabile per considerare "$\triangleright$" come un procedimento di calcolo: almeno fino al ridotto $X(YZ)$ il risultato del calcolo era unico (perchè ad ogni passo intermedio del calcolo la contrazione era unica). Si noti che la riduzione $BXYZ \triangleright X(YZ)$ non è una contrazione, ma può comunque essere usata come tale per le regole di deduzione in 2.4.

La distingueremo con "$\overset{(B)}{\triangleright}$".

Esempio. 7 *Sia* $\mathbf{C} \equiv \mathbf{S}(\mathbf{BBS})(\mathbf{KK})$ *per* X, Y, Z *generici, si ha:*

$$\begin{aligned}
\mathbf{C}XYZ \quad &\equiv \quad \underbrace{\widehat{\mathbf{S}}\widehat{(\mathbf{BBS})}\widehat{(\mathbf{KK})}\widehat{X}}_{redex}YZ \\
&\overset{(S)}{\triangleright} \quad \underbrace{\mathbf{BBS}X}_{redex}\underbrace{(\mathbf{KK}X)}_{redex}YZ \\
&\overset{(B)}{\underset{scelta}{\triangleright}} \quad \underbrace{\mathbf{B}(\mathbf{S}X)\underbrace{(\mathbf{KK}X)}_{redex}YZ}_{redex} \\
&\overset{(K)}{\underset{scelta}{\triangleright}} \quad \mathbf{B}(\mathbf{S}X)\mathbf{K}YZ \\
&\overset{(B)}{\triangleright} \quad \mathbf{S}X(\mathbf{K}Y)Z \\
&\overset{(S)}{\triangleright} \quad XZ(\mathbf{K}YZ) \\
&\overset{(K)}{\triangleright} \quad XZY;
\end{aligned}$$

che può essere in forma normale. Però contrariamente all' esempio precedente, le contrazioni scelte non erano le uniche possibili. Vediamo cosa succede cambiando le scelte:

$$\begin{aligned}
\underbrace{\mathbf{BBS}X}_{redex}\underbrace{(\mathbf{KK}X)}_{redex}YZ \quad &\overset{(K)}{\underset{scelta}{\triangleright}} \quad \mathbf{BBS}X\mathbf{K}YZ \\
&\overset{(B)}{\triangleright} \quad \mathbf{B}(\mathbf{S}X)\mathbf{K}YZ \\
&\overset{(B)}{\triangleright} \quad \mathbf{S}X(\mathbf{K}Y)Z \\
&\overset{(S)}{\triangleright} \quad XZ(\mathbf{K}YZ) \\
&\overset{(K)}{\triangleright} \quad XZY.
\end{aligned}$$

Lo stesso succederà per la seconda scelta.

Quesito: accadrà sempre che seguendo strade diverse si possa raggiungere sempre il medesimo risultato?

Esempio. 8 *Sia* $J \equiv \mathbf{SKK}$.

$$JX \equiv \mathbf{SKK}X \overset{(S)}{\rhd} \mathbf{K}X(\mathbf{K}X) \overset{(K)}{\rhd} X.$$

Cioè $JX \rhd X$ così come (**I**), quindi potremmo fare a meno della **I**.

Lemma. 2.1 *Se la variabile $x \notin X$ allora $x \notin Y$, $\forall\ Y$ tale che $X \rhd Y$.*

Lemma. 2.2 *Se $X \rhd Y$ allora $[N/X]X \rhd [N/X]Y$ e lo stesso vale per le sostituzioni multiple (la dimostrazione è abbastanza ovvia per induzione).*

Def. 2.7 *Mentre nei termini combinatori possono occorrere atomi diversi dalle costanti irriducibili, nei tre esempi notevoli precedenti solo le costanti riducibili vi occorrono. Pertanto i termini (come B, C, J) in cui atomi diversi da tali costanti non occorrono si dicono* combinatori.

2.1.4 Astrazione

Def. 2.8 (astrazione) *L'*astrazione *di un termine combinatorio M rispetto ad una variabile x indicata con $[x]M$, è definita per induzione su M da:*

a) $[x]x \equiv I$;

b) $[x]M \equiv KM, \quad$ *per* $x \notin M$;

c) $[x]Ux \equiv U, \quad$ *per* $x \notin U$;

d) $[x]UV \equiv S([x]U)([x]V), \quad$ *altrimenti.*

Esempio. 9 *Astrazione dei termini yx e xy rispetto ad x:*

- $[x]yx \overset{c)}{\equiv} y, \quad$ *(per $y \neq x$);*
- $[x]xy \overset{d)}{\equiv} S([x]x)([x]y) \overset{ab)}{\equiv} SI(Ky).$

2.1.5 Riduzione semplice

Teorema. 2.1 (riduzione semplice) *Per ogni variabile x e per tutti i termini combinatori M ed N si ha che:*

$$\big([x]M\big)N \ \rhd\ [N/x]M.$$

Dim. 2.1 (riduzione semplice) *Osserviamo dapprima che è sufficiente dimostrare l'asserto nel caso $N \equiv x$, cioè dimostrare che:*

$$\big([x]M\big)x \ \rhd\ [x/x]M \equiv M. \qquad (1)$$

Come si vede facilmente dalla definizione di sostituzione, possiamo limitarci a tale N poichè da $X \rhd Y$ segue che $[N/X]X \rhd [N/X]Y \quad \forall N$. Infatti nel nostro

caso ciò ci porterebbe all'asserto, poichè $x \notin [x]M$ *(come segue dalla definizione di astrazione) e da ciò viene che* $[N/x]\big([x]M\big)x \equiv \big([x]M\big)N$.

Per dimostrare la (1) *procediamo per induzione combinatoria su* M *raggruppando i casi in corrispondenza della definizione di astrazione.*

- *Caso* $M \equiv x$.

  $[x]M \equiv \mathbf{I}$, *per cui:*

  $([x]M)x \equiv \mathbf{I}x \overset{(I)}{\rhd} x \equiv M.$ *da cui la 1)*

- *Caso* $x \notin M$.

  $[x]M \equiv \mathbf{K}M$, *per cui:*

  $([x]M)x \equiv \mathbf{K}Mx \overset{(K)}{\rhd} M.$ *da cui la 1)*

  Esaurita così la base.

- *Caso* $M \equiv Ux$ *con* $x \notin U$.

  $[x]M \equiv U$, *per cui:*

  $([x]M)x \equiv Ux \overset{(\rho)}{\rhd} M$ *da cui la 1)*

- *Caso* $M \equiv UV$ *non già trattato, cioè* $x \in U$ *oppure* $(x \neq V$ *e* $x \in V)$*.In questo caso composto servirà l'ipotesi del passo induttivo, cioè :*

  $$([x]U)x \rhd U \text{ e } ([x]V)x \rhd V. \qquad (2)$$

  $[x]M \equiv S([x]U)([x]V)$, *per cui:*

  $([x]M)x \equiv \mathbf{S}([x]U)([x]V)x \overset{(S)}{\rhd} ([x]U)x(([x]V)x) \overset{(2)}{\equiv} UV \equiv M.$

2.1.6 Riduzione multipla

Teorema. 2.2 *Per tutte le variabili* $x', x'', \ldots, x^{(n)}$ *e per tutti i termini combinatori* $M, N', N'', \ldots, N^{(n)}$ *si ha che:*

$$([x', x'', \ldots, x^{(n)}]M)N'N'' \ldots N^{(n)} \rhd [N'/x', N''/x'', \ldots N^{(n)}/x^{(n)}]M,$$

qualora le $x^{(i)}$ *siano distinte e* $[x', x'', \ldots, x^{(n)}]$ *indichi l'astrazione iterata* $[x']([x''](\ldots [x^{(n)}]N \ldots))$.

Esempio. 10 *Purchè le variabili siano ancora distinte l'astrazione multipla può ridefinire i nostri combinatori, cioè le costanti riducibili.*

- $[x]x \equiv \mathbf{I}$;
- $[x, y]x \equiv [x]([y]x) \equiv [x]\mathbf{K}x \equiv \mathbf{K}$;

- $[x,y,z]xz(yz) \equiv [x,y]([z]xz(yz)) \equiv [x,y]\mathbf{S}([z]xz)([z]yz) \equiv [x,y](\mathbf{S}xy) \equiv$

  $\equiv [x]([y]\mathbf{S}xy) \equiv [x]\mathbf{S}x \equiv \mathbf{S}.$

La dimostrazione del teorema precedente proviene facilmente da quello per la riduzione semplice usando le seguenti osservazioni.

Oss. 2.3 *Il lemma della conservazione della sostituzione da parte della "$\triangleright$" vale anche nel caso multiplo.*

$$X \triangleright Y \quad \Longrightarrow \quad [N'/x', \ldots N^{(n)}/x^{(n)}]X \quad \triangleright \quad [N'/x', \ldots N^{(n)}/x^{(n)}]Y.$$

Lemma. 2.3 *Se $v \notin N$, allora $[v]([v/x]N) \equiv [x]N$, cioè non importa come si chiamano le variabili d'astrazione.*

Lemma. 2.4 *Se $x \not\equiv y$ e $y \notin N$, allora $[N/x]([y]M) \equiv [y]([N/x]M)$ per ogni termine N ed M.*

2.1.7 Uso del teorema di astrazione

Mentre la "$\triangleright$" ci definisce un processo di calcolo tra termini combinatori, il teorema di riduzione ci permette di trovare dei termini combinatori aventi certe proprietà di calcolo. Infatti, se vogliamo trovare un termine combinatorio X tale che:

$$XN' \ldots N^{(n)} \triangleright \underbrace{[N'/x', \ldots, N^{(n)}/x^{(n)}]Y}_{da\ pensarsi\ assegnato},$$

ci basterà per il teorema di riduzione nel caso multiplo calcolare:

$$[x', \ldots, x^{(n)}]Y \equiv X.$$

Esempio. 11 *Cerchiamo un X tale che per ogni u, v e z:*

$$XUVZ \triangleright U(VZ) \equiv [U/u, V/v, Z/z]\underbrace{u(vz)}_{Y}.$$

Possiamo trovare un tale X come $[u,v,z]u(vz)$. abbiamo così:

$$\begin{array}{rcl} [u,v,z]u(vz) & \equiv & [u,v]([z]u(vz)) \\ & \equiv & [u,v](\mathbf{S}([z]u)([z]vz)) \\ & \equiv & [u,v]\mathbf{S}(\mathbf{K}u)v \\ & \equiv & [u]([v]\mathbf{S}(\mathbf{K}u)v) \\ & \equiv & [u]\mathbf{S}(\mathbf{K}u) \\ & \equiv & \mathbf{S}([u]\mathbf{S})([u]\mathbf{K}u) \\ & \equiv & \mathbf{S}(\mathbf{KS})\mathbf{K}. \end{array}$$

Che è il nostro $\mathbf{B}$.

2.2 Proprietà di riduzione ed uguaglianza debole

Teorema. 2.3 (Church-Rosser I) *Per ogni termine combinatorio X, Y, Z se X si riduce a Y $(X \rhd Y)$ e X si riduce a Z $(X \rhd Z)$, allora esiste un termine combinatorio U tale che $Y \rhd U$ e $Z \rhd U$*

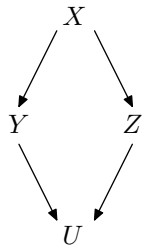


Figura 2.9: Il calcolo anche se segue strade diverse può convergere.

Dim. 2.2 *No. (induzione)*

(Il calcolo, anche se segue strade diverse, può convergere. Si parla di proprietà di Church-Rosser per relazioni come la nostra "$\rhd$", non per funzioni)

Oss. 2.4 *Si noti che U non è detto che sia in forma normale. Quindi tale proprietà esprime un'unicità di "risultato", anche quando il risultato non c'è.*

Corollario 2.1 *L'eventuale forma normale di un termine combinatorio è* unica.

Dim. 2.3 *Supponiamo che $X \rhd Y, Z$ ove Y e Z sono in forma normale. Poichè esiste U tale che $Y, Z \rhd U$ e poichè Y, Z non hanno alcun redex, tali ultime riduzioni non sono contrazioni, cioè provengono dall'assioma di riflessività (ρ). Pertanto $Y \equiv U \equiv Z$ cioè per trasitività $Y \equiv Z$.*

Def. 2.9 *Diremo che X è* (debolmente) uguale*, e scriveremo $X = Y$ qualora ciò si possa dedurre come nella definizione di riduzione (con $=$ al posto di $\rhd$) con l'aggiunta della regola di deduzione di simmetria:*

$$X = Y \Longrightarrow Y = X$$

Così facendo si intuisce subito che l'uguaglianza debole è la chiusura simmetrica della riduzione. Si può anche facilmente vedere che:

- *Se $X \rhd Y$ allora esiste una catena finita di contrazioni che porta da X a Y, cioè esistono $X', X'', \ldots X^{(n)}$ tali che $X \equiv X' \rhd, \ldots \rhd X^{(n)} \equiv Y$, ove le "$\rhd$" sono contrazioni.*

- *Se $X = Y$ allora esiste una catena finita di contrazioni e contrazioni inverse, che porta da X a Y come sopra.*

Teorema. 2.4 (Church-Rosser II) *Se $X = Y$ allora esiste Z tale che $X \rhd Z$ e $Y \rhd Z$.*

Dim. 2.4 *Consideriamo una catena di contrazioni e contrazioni inverse da X a Y, come nel precedente lemma, di lunghezza m e procediamo per induzione aritmetica su $n = m + 1$.*

- *$n = 0$, non ci sono contrazioni $X \rhd Y$ nè $Y \rhd X$, per l'asserto (ρ) $X \equiv Y$. Pertanto basta prendere $Z \equiv X, Y$ e usare (ρ).*
- *Supponiamo l'asserto vero per catene di lunghezza n e consideriamo una catena di lunghezza $n+1$. Avremo allora che esiste uno Z' tale che $X \rhd Z'$ e $X^{(n)} \rhd Z'$ mentre potranno esserci due casi: $X^{(n)} \overset{contraz.}{\rhd} Y$, $Y \overset{contraz.}{\rhd} X^{(n)}$.*

- *Caso $X^{(n)} \rhd Y$ si ha:*

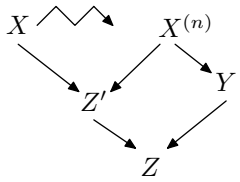


Figura 2.10: Le frecce indicano contrazioni (dirette ed inverse).

Pertanto per transitività della $\rhd$ si ha l'asserto.

- *Caso $Y \rhd X^{(n)}$, basta solo la transitività:*

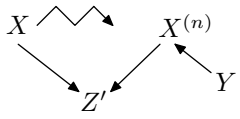


Figura 2.11: È sufficiente prendere $Z \equiv Z'$.

Corollario 2.2 (A) *Se $X = Y$ e Y è in forma normale, allora $X \rhd Y$.*

Dim. 2.5 *Per il teorema di* Church-Rosser II *esiste Z tale che $X, Y \rhd Z$. Nella $Y \rhd Z$ non ci può essere contrazione, cioè si usa (ρ).*

Corollario 2.3 (B) *Se $X = Y$ o X e Y non hanno forma normale oppure hanno la stessa forma normale.*

Dim. 2.6 *Consideriamo la prima alternativa e procediamo per assurdo, cioè supponiamo che X abbia forma normale mentre Y non l'abbia (o viceversa).*

Sia Z la forma normale di X allora per corollario A si ha che Z è forma normale anche di Y poichè $Z \lhd X = Y$. Pertanto o vale la prima alternativa o c'è questa comune forma normale che (per Corollario di Church-Rosser I) è unica.

Corollario 2.4 (C) *Se X e Y sono in forma normale, allora il fatto che $X \not\equiv Y$ implica che $X \neq Y$.*

Dim. 2.7 *Supponiamo per assurdo che $X = Y$. Applichiamo il Corollario B, cioè otterremmo che $X \equiv Y$ che è assurdo.*

(Corollario di "consistenza" logica del calcolo dei combinatori relativamente all'uguaglianza debole. Infatti esistono ovvi termini combinatori in forma normale che non sono identici.)

2.3 Interpretazione "utile" del calcolo dei combinatori

L'interpretazione "utile" considera i termini combinatori come delle funzioni insiemistiche, contrariamente al fatto che non lo possono essere, come visto nella diversità per combinatore identità e classe (insieme grande) identità.

2.3.1 Richiami sulla teoria degli insiemi

Nella formulazione del testo consigliato di Teoria degli Insiemi le funzioni sono particolari insiemi e gli insiemi delle classi costruite con degli assiomi a partire dallo zero (nei combinatori abbiamo invece parecchi atomi).

Che $\emptyset$ sia l'unico insieme atomico è una conseguenza dell'assioma di estensionalità (che dice che due insiemi, non importa come costruiti, sono lo stesso insieme se e solo se hanno gli stessi elementi).

Fra i primi insiemi definibili ci sono le coppie $\langle a, b\rangle$. Insiemi di coppie sono detti relazioni. Da una relazione r si definiscono il suo dominio:

$$\mathrm{Dom}\, R = \left\{\, a \mid \langle a, b\rangle \ \in \ r \ \right\};$$

e la controimmagine del dominio $\left\{\, b \mid \langle a, b\rangle \ \in \ r \ \right\}$ (tali oggetti sono definiti *dopo* la relazione r).

Una relazione si dice funzione se soddisfa la condizione di funzionalità:

$$\langle a, b'\rangle,\ \langle a, b''\rangle \in f \Longrightarrow b' = b''.$$

La scritta $f\colon A \to B$ significa che f è una funzione, che A è il suo dominio ($\mathrm{Dom}\, f$) e che $B \supseteq \left\{\, b \mid \ \langle a, b\rangle \in \ f \,\right\}$ cioè che gli "argomenti" di f costituiscono A e che i suoi "valori" stanno in B.

Con r^{-1} si intende la conversa di una relazione cioè $\{ \langle b, a\rangle \mid \langle a, b\rangle \in r \}$. Se per una funzione $f\colon A \to B$ si ha che f^{-1} sia funzione scriviamo $f : A \rightarrowtail B$. Se B è la controimmagine del dominio scriviamo $f : A \twoheadrightarrow B$ (suriezione su B) e se vogliamo entrambe scriviamo $f : A \rightarrowtail\!\!\!\!\twoheadrightarrow B$ (biiezione da A su B).

2.3.2 "Tipi" delle funzioni insiemistiche

L'asserzione $f : A \longrightarrow B$ serve per definire l'insieme *potenza fra due insiemi*:

$$B^A = \{f \mid f\colon A \to B\}.$$

pertanto $f : A \longrightarrow B$ sse $f \in B^A$.

Nota: Sul testo consigliato si usa mettere l'esponente sulla sinistra ${}^A\mathrm{B}$ poichè conserva l'ordinamento.

Vi sono poi altre costruzioni, per esempio il *prodotto* di una funzione $A : I \longrightarrow B$.

$$\prod_{i\in I} A_i = \left\{ f \mid f : I \longrightarrow \bigcup_{i\in I} A_i \ e\ f_i \in A_i \right\}$$

Siccome questo prodotto è un sottoinsieme della potenza la cui base è $\bigcup_{i\in I} A_i \subseteq \cup B$

$$\prod_{i\in I} A_i \subseteq \left(\bigcup_{i\in I} A_i \right)^I \subseteq (\cup B)^I ,$$

questa costruzione è secondaria. Noi consideremo principalmente la potenza. Iterando le potenze si possono ottenere strutture di potenze: $(C^B)^A$, C^{B^A}, ...

Esempio. 12

$$f(x) = \int_a^x ,$$

inteso come funzione della funzione itegranda g, è una funzione f tale che:

$$f \in (R^R)^{R^R},$$

l'esponente al di fuori delle parentesi è dovuto alla g tale che $\int_a^x g(y)\,\mathrm{d}x = f(x)$, ovvero $f \in (R^{R^R})^R$ se si considera la x come prima variabile. Pertanto avremo a che fare con delle funzioni $F : A \longrightarrow B$ dove A e B saranno costruzioni insiemistiche complesse, fra cui tali potenze iterate.

Diremo che la parte destra dell'asserzione $F : A \longrightarrow B$ è il tipo insiemistico di F. Quindi il tipo è una potenza in cui base ed esponente possono essere complessi.

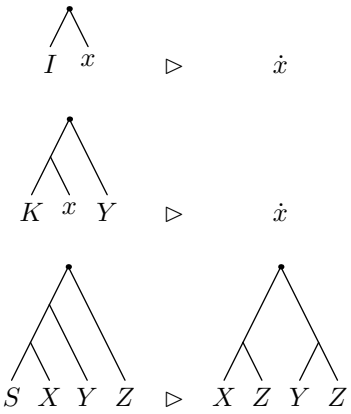


Figura 2.12: Interpretazione "arborea" delle costanti riducibili.

2.3.3 Interpretazione Insiemistica delle costanti riducibili

L'interpretazione "arborea" finora considerata è quella corretta. L'interpretazione utile è invece quella che vede negli **I**, **K**, **S** delle funzioni insiemistiche che partono sempre dagli assiomi riferiti all'uguaglianza debole vista come uguaglianza insiemistica.

$\mathbf{I}X = X$ mi dice che **I** è identità, possiamo (sbagliando) pensarla come identità sulla classe V di tutti gli insiemi, allora: $\mathbf{I}\colon V \to V$ oppure come identità su un insieme qualsiasi $\mathbf{I}\colon A \to A$

Si noti che nel secondo caso si ha anche che $\mathbf{I}\colon A \to B$ per ogni $B \supseteq A$.

Consideriamo $\mathbf{K}XY = X$, insiemisticamente ciò ci dice che abbiamo una funzione $\mathbf{K}_X$ tale che $\mathbf{K}_X(Y) = X$. Quindi tale $\mathbf{K}_X$ è una funzione costante e $\mathbf{K}_X : A \longrightarrow B$ ove $A \ni Y$ e $B \ni X$.

Pertanto con tali A e B si ha che $\mathbf{K}\colon B \to B^A$ e K ha come valori delle costanti, in definitiva K è insiemisticamente un generatore di costanti.

Consideriamo $\mathbf{S}XYZ = XZ(YS)$, aiutiamoci con la scrittura delle applicazioni insiemistiche:

$$[\mathbf{S}(X)]_Y(Z) = X_Z(Y_Z).$$

Quindi considerato $Y_Z\ Y : C \longrightarrow B$ ove $C \ni Z$ e $X_Z\colon B \to A$, da cui $X\colon C \longrightarrow A^B$.

Esempio. 13 *Definiamo i sopracitati insiemi C, X e Y nel seguente modo:*

$C = \{0, 1, 2, \}$ $Y =$

Z	Y_Z
0	*5*
1	*4*
2	*2*

$X =$

Z	X_Z
0	*successore*
1	$\sqrt{\ }$
2	*dimezzamento*

Cosicchè X_ZY_Z vale 6, 2 *o* 1 *a seconda che Z sia* 0, 1 *o* 2. *In sostanza Z è un indice per dei vettori X e Y. Il vettore X è un vettore di funzioni (come operazioni unarie) e Y lo è di numeri.*

X_Z	*successore*	$\sqrt{}$	*dimezzamento*
Y_Z	5	4	2
X_ZY_Z	6	2	1

L'ultimo vettore dei risultati è stato ottenuto applicando "in parallelo" Y ad X. Tale vettore corrisponderà a SXY poichè $(SXY)Z = XZ(YZ)$ ovvero riscrivendo: $([S(X)]_Y)(Z) = X_Z(Y_Z)$.
Analogamente SX overo $S(X)$ corrisponderà all'operatore che consente tale operazione parallela ad X di un vettore di argomenti qualsiasi poichè $[S(X)]_y$ con y variabile fà ciò.

Ma allora S non è altro che l'operatore parallelizzazione *che da X ci da il precedente $S(X)$ per X vettore di operazioni unarie qualsiasi.*

Osserviamo ora i tipi:

$$\text{Per le notazioni precedenti } \begin{cases} Z \in C; \\ Y : C \longrightarrow B; \\ X : C \longrightarrow A^B. \end{cases}$$

Se consideriamo $[S(X)]_Y : C \longrightarrow A$, poichè $[S(X)]_Y(Z) = XZ(YZ)$, allora:

$$S(X) : B^C \longrightarrow A^C \text{ poichè } [S(X)]_Y : C \longrightarrow A \text{ e } Y \in B^C$$

e

$$S : (A^B)^C \longrightarrow (A^C)^{B^C} \text{ poichè } X \in (A^B)^C \text{ e } S(X) \in (A^C)^{B^C}.$$

Quanto fatto ora può essere ripetuto per altri termini.

Consideriamo $\mathbf{B}$ e $\mathbf{B}XYZ = X(YZ)$. Sia $A \ni Z$ e $Y\colon A \to B$ ove B sia il dominio di X. Pertanto detto C un insieme che contenga i valori di X si ha che X sarà funzione che va da B in C ($X\colon B \to C$). Per la nostra relazione di uguaglianza "=" $\mathbf{B}XY$ non è altro che la composizione funzionale di Y ed X, cioè $\mathbf{B}XY = X \circ Y$ e $\mathbf{B}XY\colon A \to C$.

Pertanto $\mathbf{B}X$ è l'operatore che compone una funzione arbitraria Y con X e si pensa che $\mathbf{B}X\colon B^A \to C^A$ poichè $Y \in B^A$ e $BXY \in C^A$, in definitiva si pensa a:

$$B : C^B \longrightarrow (C^A)^{B^A};$$

il quale è un operatore di composizione funzionale invertito (in $BXY = X \circ Y$ il primo è Y anche se si scrive $(X \circ Y)(Z) = (Y(Z))$. "Ragione storica" per notazione convenzionale dell'applicazione insiemistica nata tre secoli fa.

2.4 Teorema del punto fisso

Ricordiamo che una funzione insiemistica f si dice abbia un punto fisso qualora esista nel suo dominio un a tale che $f(a) = a$. Le funzioni insiemistiche possono comunque non avere punti fissi; per esempio nel campo reale $f(x) = x + 1$ ci da una funzione che ne è priva.

Teorema. 2.5 (punto fisso) *Ogni termine combinatorio (rispetto all'applicazione combinatoria) X ha un punto fisso poiché esiste un termine combinatorio Y, detto* combinatore di punto fisso*, i cui valori sono punti fissi per X qualunque X si scelga.*

$$X(YX) = YX.$$

Dim. 2.8 (punto fisso) *Definiamo Y come segue:*

$$Y = [x]\big([y]x(yy)\big)\big([y]x(yy)\big).$$

Allora dal teorema di riduzione si ha:

$$\begin{array}{lcl} YX & \rhd & [y]\underbrace{X(yy)}_{M}\underbrace{\big([y]X(yy)\big)}_{N} \\ & \rhd & X\Big(\big([y]X(yy)\big)\big([y]X(yy)\big)\Big) \\ & \lhd & X\Big(\underbrace{[x]\big([y]x(yy)\big)\big([y]x(yy)\big)}_{Y}X\Big) \\ & \equiv & X(YX). \end{array}$$

Abbiamo supposto per semplicità che $x, y \notin X$, si vedano i lemmi 2.3 e 2.4.

Oss. 2.5 *Le riduzioni dirette (o inverse) considerate comportano delle contrazioni. Quindi YX non può essere in forma normale, ma anche $X(YX)$ non può esserlo:*

$$YX = X(YX) = X\big(X(YX)\big) = \ldots$$

Quindi abbiamo dei termini combinatori che non possono avere forma normale, per esempio ciò accade prendendo per X una variabile. Talvolta tale Y ẽ servito per implementare la ricorsione nei linguaggi di programmazione.

Oss. 2.6 *L'uguaglianza debole non conserva l'astrazione e non vale la proprietà di estensionalità (est).*

a) L'uguaglianza debole non conserva l'astrazione, cioè:

$$X = Y \Longrightarrow [x]X = [x]Y, \quad \textit{(in generale).}$$

Controesempio:

$$X \equiv \mathbf{S}xyz,\ Y \equiv xz(yz),\ \textit{per (S) e } X = Y,\ \textit{ma:}$$

$$[x]X \equiv \mathbf{S}\big([x]\mathbf{S}xy\big)\big([x]z\big) \equiv \mathbf{S}(\mathbf{SS}(\mathbf{K}y))(\mathbf{K}z);$$

$$[x]Y \equiv \mathbf{S}\big([x]xz\big)\big([x]yz\big) \equiv \mathbf{S}\big(\mathbf{SI}(\mathbf{K}z)\big)\big(\mathbf{S}(\mathbf{K}y)(\mathbf{K}z)\big).$$

Sia $[x]X$ che $[x]Y$ sono in forma normale, ma non sono equivalenti e per il corollario del teorema di Church-Rosser devono essere non debolmente uguali ($\neq$).

b) Non vale la proprietà di estensionalità:

$$XZ = YZ \text{ per ogni termine combinatorio } Z \text{ allora } X = Y; \qquad (est)$$

ciò si può vedere dal controesempio precedente, poichè per $X' \equiv [x]X$ e $Y' \equiv [x]Y$:

$$\begin{aligned} X'Z &= SZyz \\ &= Zz(yz) \\ &\equiv [Z/x]xz(yz) \\ &= Y'Z. \end{aligned}$$

La proprietà (est) è detta di estensionalità*. Infatti essa corrisponde a quella di estensionalità delle funzioni insiemistiche (che deriva dall'assioma di estensionalità degli insiemi) che asserisce:*

$$f(a) = g(a) \ \forall \ a \Longrightarrow f = g$$

Si noti che per i termini combinatori l'uguaglianza debole deriva dalla riduzione debole che rappresenta un processo di calcolo idealizzato su un programma completo (dei suoi dati).

2.5 Esercizi significativi

In questa sezione del capitolo mostreremo due esercizi importanti svolti in classe con la relativa interpretazione funzionale.

2.5.1 Diagonalizzatore

Partendo dal seguente esercizio otterremo un'operatore che dato un termine combinatorio X, visto come una matrice, ne trova la diagonale.

Esercizio. 2 *Trovare un termine combinatorio* $\mathbf{W}$ *tale che:*

$$\mathbf{W}XY \rhd XYY.$$

Quindi consideriamo: $\mathbf{W}xy \rhd xyy$.

$$\begin{aligned} \mathbf{W} &\equiv [x,y]xyy \\ &\equiv [x]\big([y]xyy\big) \\ &\equiv [x]\Big(\mathbf{S}\big([y]xy\big)\big([y]y\big)\Big) \\ &\equiv [x](\mathbf{S}x\mathbf{I}) \\ &\equiv \mathbf{S}\big([x]\mathbf{S}x\big)\big([x]\mathbf{I}\big) \\ &\equiv \mathbf{SS}(\mathbf{KI}). \end{aligned}$$

Applichiamo ora tale $\mathbf{W} \equiv \mathbf{SS}(\mathbf{KI})$ alle variabili richieste xy:

$$\begin{array}{rcl} \mathbf{W}xy & \equiv & \mathbf{SS}(\mathbf{KI})xy \\ & \rhd & \mathbf{S}x(\mathbf{KI}x)y \\ & \rhd & \mathbf{S}x\mathbf{I}y \\ & \rhd & xy(\mathbf{I}y) \\ & \rhd & xyy. \end{array}$$

Questo dimostra che il termine combinatorio $\mathbf{W}$ sopra definito è corretto.

2.5.2 Interpretazione funzionale

Partendo dal secondo membro (xyy) si nota che x è una funzione i cui argomenti sono pensabili come $y \in A$ per qualche A. Ma i suoi valori (xy) sono a loro volta funzioni i cui argomenti sono contenuti in un insieme B.

Allora xy è da pensare come $x_y \colon A \to B$. Pertanto:

$$x \colon A \to B^A.$$

Per esempio se consideriamo una matrice quadrata X (nel senso di array) vista per colonne, potremmo pensare a B come ad un insieme che contenga gli elementi delle colonne.

Se noi uguagliamo xyy a Wxy possiamo ora considerare il primo membro Wxy come il valore in y (colonna o riga y-esima) di una funzione Wx che è definita da X. Tale Wx (o W_x) restituisce allora $x_y(y)$, cioè il y-esimo elemento della y-esima colonna (o riga) di X.

Perciò W_x è la funzione che ci da la diagonale della matrice X. Potremo quindi dire che W è un diagonalizzatore.

2.5.3 Estrattore parametrico

Nel precedente esercizio $\mathbf{W}$ estraeva da una matrice il "percorso" della sua diagonale, cioè quello definito dai parametri U, $V \equiv \mathbf{I}$ per cui $Uy = y$ e $Vy = y$ individuano un elemento di diagonale. Partendo dal seguente esercizio otterremo un'operatore che dato un termine combinatorio X visto come una matrice, estrae il percorso definito parametricamente da U e V arbitrariamente (con indice Y).

Esercizio. 3 *Trovare un termine combinatorio* Φ *tale che:*

$$\Phi XUVY \rhd X(UY)(VY).$$

Quindi consideriamo: $\Phi xuvy \rhd x(uy)(vy)$.

$$
\begin{aligned}
\Phi &\equiv [x,u,v,y]x(uy)(vy)\\
&\equiv [x,u,v]\big([y]x(uy)(vy)\big)\\
&\equiv [x,u,v]\Big(\mathbf{S}\big([y]x(uy)\big)\big([y]vy\big)\Big)\\
&\equiv [x,u,v]\Big(\mathbf{S}\big(\mathbf{S}([y]x)([y]uy)\big)v\Big)\\
&\equiv [x,u,v]\Big(\mathbf{S}\big(\mathbf{S}(\mathbf{K}x)u\big)v\Big)\\
&\equiv [x,u]\Big([v]\mathbf{S}\big(\mathbf{S}(\mathbf{K}x)u\big)v\Big)\\
&\equiv [x,u]\mathbf{S}\big(\mathbf{S}(\mathbf{K}x)u\big)\\
&\equiv [x]\Big([u]\mathbf{S}\big(\mathbf{S}(\mathbf{K}x)u\big)\Big)\\
&\equiv [x]\mathbf{S}\big(\mathbf{S}(\mathbf{K}x)\big)\\
&\equiv \mathbf{S}([x]\mathbf{S})\Big([x]\big(\mathbf{S}(\mathbf{K}x)\big)\Big)\\
&\equiv \mathbf{S}(\mathbf{KS})\big(\mathbf{S}([x]\mathbf{S})([x]\mathbf{K}x)\big)\\
&\equiv \mathbf{S}(\mathbf{KS})\big(\mathbf{S}(\mathbf{KS})\mathbf{K}\big).
\end{aligned}
$$

2.5.4 Interpretazione funzionale

Φ differisce da **W** solo perchè in luogo di U e V (arbitrarie) usiamo le **I** (identità). Ma allora specificando diversamente U e V si da un percorso differente dalla diagonale. Il percorso è "specificato parametricamente" da U e V mentre Y è sempre l'indice d "percorrenza". Φ è quindi un "estrattore parametrico".

2.5.5 Conclusioni

Con le costanti riducibili (le primitive del linguaggio di programmazione idealizzato) **S** e **K** siamo stati in grado di costruire altri operatori (diversi dal generatori di costanti e dal parallelizzatore):

- **B**: operatore di composizione invertito;
- **C**: invertitore, che come vedremo sotto generalizza la trasposizione;
- **W**: diagonalizzatore;
- Φ: estrattore parametrico.

Da $\mathbf{C}XYZ \rhd XZY$ con il ragionamento visto per **W** si ha che, pensando ad X come matrice per colonne, $\mathbf{C}X$ è la stessa mastrice ma per righe. Cioè **C** è la generalizzazione della trasposizione matriciale. Si noti anche che il nome è necessario e che la matrice X sia pensata quadrata.

Con i soliti riferimenti insiemistici si può vedere la **C** come:

$$\mathbf{C}\colon (A^B)^C \to (A^C)^B.$$

Ci si può allora chiedere che cosa d'altro le nostre primitive consentano di calcolare.

Capitolo 3

Funzioni ricorsive

Secondo la tesi di Church (e l'equivalente tesi di Turing), tutto ciò che è calcolabile può essere definito come tutto ciò che ci viene dato da qualche funzione parziale ricorsiva.

3.1 Funzioni parziali ricorsive

Una funzione fra numeri naturali si dice parziale ricorsiva quando può essere ottenuta da delle funzioni base mediante due operatori.

Funzioni di base :

- costante 0;
- somma;
- proiezioni $U_i^n(x_0, \ldots, x_{n-1}) = x_i$.

Operatori :

- Composizione multipla;

  data una funzione f ad n argomenti ed n funzioni g_i ad m argomenti, otteniamo una funzione h ad n argomenti tale che:

  $$h(x_0, \ldots, x_{n-1}) = f\big(g_0(x_0, \ldots, x_{m-1}), \ldots, g_{n-1}(x_0, \ldots, x_{m-1})\big).$$

- Minimalizzazione;

  data una funzione g ad $n+1$ argomenti otteniamo una funzione h ad n argomenti tale che: $h(x_0, \ldots, x_{n-1}) = \min y$ per il quale $g(y, x_0, \ldots, x_{n-1}) = 0$ e si scrive:

  $$h(x_0, \ldots, x_{n-1}) = \mu_y\big[g(y, x_0, \ldots, x_{n-1}) = 0\big].$$

In realtà l'operatore μ è un operatore d'inversione.

Esempio. 14 *Vediamo una possibile applicazione dell'operatore μ:*

$$\mu_y(x_o y - x_1 = 0) = \frac{x_1}{x_0}$$

cioè μ ha invertito la moltiplicazione.

Per certe operazioni vi possono essere più risultati d'inversione.

Esempio. 15 *L'equazione:*

$$(y-2)^2 - x_0 = 0$$

per $x_0 = 4$ restituisce sia 0 che 4, al fine di ottenere un solo risultato si sceglie il minimo.

3.2 Alcune considerazioni

Che la classe delle funzioni parziali ricorsive possa essere una candidata per la definizione della classe delle funzioni calcolabili segue dalle seguenti considerazioni:

A le funzioni di base sono notoriamente calcolabili,

B le costruzioni date dagli operatori di composizione multipla e minimizzazione (μ) conservano la calcolabilità[1].

Tale classe è detto di funzioni parziali perchè la μ anche nel caso di g ovunque definita può dare una fuzione che non è ovunque definita (esempio $\mu_y(2y-x=0)$ è una funzione di x definita solo per gli x pari).

Tali funzioni si dicono semplicemente *ricorsive* quando sono "totali", cioè definite per qualunque scelta degli argomenti. In generale le funzioni parziali ricorsive sono dette "ricorsive" perchè si può dimostrare che con gli operatori funzionali e funzioni di base si possono generare funzioni definite tramite delle ricorsioni.

[1]La composizione multipla fra funzioni calcolabili ci da sempre una funzione calcolabile mediante il "passaggio" dei valori delle g_i agli argomenti della f. Anche dalla μ otteniamo funzioni calcolabili, basta eseguire le seguenti procedure:

a) $y = 0$;

b) calcola $g(y, x_0, \ldots)$;

c) verifica se $g(y, x_0, \ldots) = 0$, se si esci;

d) altrimenti incrementa y e vai a b).

Si può verificare che esiste una procedura equivalente, sia pure più complicata, anche per il caso che qualche valore di g non sia definito.

3.3 Ricorsione primitiva

Vi sono parecchi casi di ricorsione. Uno dei più semplici ed apparentemente potente è quello della *ricorsione primitiva*.

Def. 3.1 *Una funzione h di $n+1$ argomenti si dice ottenuta per* ricorsione primitiva *da una funzione g ad n argomenti e da una funzione f a $n+2$ argomenti se:*

$$\begin{cases} h(0, x_0, \ldots, x_{n-1}) = g(x_0, \ldots, x_{n-1}); \\ h(k+1, x_0, \ldots, x_{n-1}) = f\big(k, h(k, x_0, \ldots, x_{n-1}), x_0, \ldots, x_{n-1}\big). \end{cases}$$

Si noti che se la f e la g sono totali, tali equazioni definiscono una h che è totale a sua volta e conserva ovviamente la calcolabilità.

Oss. 3.1 *Non tutte le costruzioni ricorsive sono primitive*

Esempio. 16 *Le due seguenti condizioni (ove $\frac{k}{2}$ indica il quoziente di k diviso 2) ci definiscono una l ad un argomento che è totale.*

$$l(0) = 0$$

$$l(k+1) = l\left(\frac{k}{2}\right) + 1$$

Infatti da esse si ha:

n	$l(n)$
0	0
1	1
2	1
3	2
4	2
⋮	⋮
7	3
8	3
⋮	⋮
15	4

Cioè "quasi un logaritmo".

Tuttavia la costruzione della l non *è ricorsiva primitiva. Infatti il passo di ricorsione è diverso (usa un valore "$\frac{k}{2}$" differente dal predecessore per calcolare $l(k+1)$).*

3.4 Funzioni ricorsive primitive

Usando opportune funzioni di base (totali): la precedente composizione multipla e tale ricorsione primitiva, si può definire un'ampia sottoclasse delle funzioni parziali ricorsive dette *funzioni primitive ricorsive*. Per quanto osservato tali funzioni sono totali. Si vede facilmente che fra di esse vi sono:

$h(x, y) = x + y$;

$h(x, y) = xy$;

$h(x, y) = x^y$;

$$h(x, y) = \begin{cases} 0 & \text{se } x < y; \\ x - y & \text{altrimenti;} \end{cases}$$

etc.

Tuttavia esistono funzioni chiaramente calcolabili, anche totali, che non sono primitive ricorsive, cioè tali che non puó esistere alcuna costruzione di esse nella classe delle funzioni primitive ricorsive. Una di esse è la funzione di Ackerman.

Tale funzione in due variabili: $ack(x, y)$ al crescere di y restituisce delle funzioni in x come le seguenti:

$\vdots$
x
x^2
$x^x \ldots$
$\vdots$
(superesponenziale)

Essa è totale ed esiste una definizione ricorsiva (non primitiva) abbastanza semplice per tale *ack*.

(Si noti che per la funzione l vista in precedenza l'esistenza della costruzione ricorsiva non primitiva vista *non* è sufficiente per affermare che tale funzione non sia primitiva.)

3.5 Lemma di parzialità

Si potrebbe pensare che possa esistere una classe così ampia di funzioni totali (che contenga ad esempio *ack*) e costruibile senza costruzioni (come la μ), che non conservano la totalità, da contenere ogni funzione "ragionevolemente calcolabile". Tuttavia esiste un lemma che, definendo opportunamente delle condizioni di ragionevolezza di calcolabilità, mostra la costruzione della classe necessita di operatori (come μ) che non conservano la totalità.

3.6 Tesi di Church

La tesi di Church afferma che la definizione delle funzioni fra numeri naturali, che formalizza nel modo più ampio possibile la nozione di calcolabilità, è quella della classe delle funzioni parziali ricorsive. Pertanto tale classe non è solo una candidata ma, a meno di equivalenze, la soluzione del problema definitorio.

La giustificazione di tale tesi consiste nelle seguenti asserzioni:

- *Sufficienza puntuale.* Non è mai stata trovata una funzione che si fosse in grado di calcolare per la quale si potesse dimostrare che non era nella nostra classe (per esempio *ack*, anche se definita con una ricorsione senza minimalizzazione, è dimostrato che sia definibile con le costruzioni della nostra classe).

- *Sufficienza di classe.* Tutte le classi notevoli di funzioni sinora scoperte sono state dimostrate essere contenute nella nostra classe. In particolare si è dimostrata l'uguaglianza con classi di funzioni la cui generalità di calcolabilità era intuibile da considerazioni differenti (ad esempio la classe delle funzioni calcolabili secondo Turing).

3.7 Forma normale delle funzioni parziali ricorsive

Secondo la tesi di Church funzione calcolabile, cioè parziale ricorsiva, può essere definita a partire dalle funzioni di base (0, +, U_i^n) relative mediante un uso iterato degli operatori di composizione (multipla) e μ. Il seguente teorema ci dice che per qualunque funzione parziale ricorsiva basta una sola μ, a patto di usare delle funzioni primitive ricorsive.

Quest'ultime sono definite a partire da funzioni di base come le precedenti, ma con la somma sostituita dal *successore* da degli operatori che conservano la totalità delle funzioni (non c'è la μ). Le funzioni parziali ricorsive pertanto sono realizzabili da programmi che danno sempre il risultato della funzione. L'operatore cirico (μ), che invece produce funzioni parziali, potrebbe essere usato una volta sola.

Teorema. 3.1 *Per ogni funzione parziale ricorsiva φ ad n argomenti esistono due funzioni primitive ricorsive f ad un argomento e g a $n+1$ argomenti tale che per tutte le n-uple $(x_0, \ldots, x_{n-1})$ nel suo dominio si ha:*

$$\varphi(x_0, \ldots, x_{n-1}) = f\Big(\mu_y\big(g(y, x_0, \ldots, x_{n-1}) = 0\big)\Big).$$

Dim. 3.1 *No.*

Il teorema può essere dimostrato mediante la macchina di Turing. Si formalizza tramite delle funzioni primtive ricorsive il funzionamento di una macchina che inizia a calcolare con dati iniziali $(x_0, \ldots, x_{n-1})$. Le transizioni di configurazioni della macchina di Turing corrispondono sostanzialmente al parametro y e la funzione primitiva ricorsiva g rappresenta, quando ha valore nullo, l'arresto della macchina. Dalla configurazione corrispondente a tale y "nullificante" la f estrae il risultato.

Capitolo 4

Rappresentazione combinatoria delle funzioni parziali ricorsive

Il calcolo dei combinatori (e il λ-calcolo) sono nati anche come possibile fondazione della matematica, alternativa alla teoria degli insimi. Siccome la seconda è stata sviluppata precedentemente, non è stata sviluppata (completamente) la prima. Tuttavia esiste la dimostrazione che tale calcolo avrebbe potuto diventare una fondazione alternativa. Tale dimostrazione consiste nel mostrare come all'interno del calcolo dei combinatori si possano rapresentare le funzioni parziali ricorsive. Per vedere ciò occorre dapprima rappresentare i numeri naturali.

Vi sono parecchi modi per rappresentare i numeri naturali con termini combinatori, quello che vedremo consiste in una rappresentazione "naturale" mediante soli combinatori, detti *iteratori di Church*. Essi catturano il significato *operativo* di un numero naturale appunto come "iteratore". Ciò sarà dimostrato dal primo lemma seguente.

Notazione. 1 *Conveniamo di indicare con* X^nY *ove* $n \in \mathbb{N}$ *i termini seguenti:*

$$X^0Y \equiv Y;$$

$$X^{n+1}Y \equiv \underbrace{X(X(\ldots(X}_{n+1}Y)\ldots)).$$

Definiamo per ogni numero naturale n *il suo* rappresentante combinatorio *come il combinatore* $\overline{n}$ *tale che:*

$$\overline{0} \equiv KI;$$

$$\overline{n+1} \equiv (SB)^{n+1}(KI) \text{ ovvero } \overline{n+1} = SB(\overline{n}).$$

Lemma. 4.1 *Per ogni termine combinatorio F e Y si ha che:*

$$\overline{n}FY = F^nY \text{ o meglio } \overline{n}FY \rhd F^nY.$$

Dim. 4.1 *Per induzione su n.*

- *Base $n = 0$.*
  $$\begin{array}{rcl} \overline{0}FY & \equiv & KIFY \\ & \rhd & IY \\ & \rhd & Y \\ & \equiv & F^0Y. \end{array}$$
- *Ipotesi induttiva $\overline{n}FY \rhd F^nY$.*
- *Passo induttivo:*
  $$\begin{array}{rcl} \overline{n+1}FY & \equiv & SB\overline{n}FY \\ & \rhd & BF(\overline{n}F)Y \\ & \rhd & F(\overline{n}FY) \\ & \rhd & F(F^nY) \\ & \equiv & F^{n+1}Y. \end{array}$$

Def. 4.1 *Data una funzione tra numeri naturali φ ad n argomenti diremo che un termine combinatorio X la* rappresenta combinatoriamente (in senso debole) *qualora:*

$$X\overline{x}_0\overline{x}_1 \dots \overline{x}_{n-1} = \overline{\varphi(x_0, x_1, \dots, x_{n-1})} \quad \forall\ x_0, x_1, \dots, x_{n-1} \in \mathrm{Dom}\,\varphi.$$

Se inoltre succede che per le variabili $x_0, x_1, \dots, x_{n-1}$ fuori dal dominio di φ il primo membro non ha forma normale diremo che X la rappresenta combinatoriamente *in senso forte*. Le rappresentazioni in senso forte non sono però necessarie per noi.

Oss. 4.1 *Si noti che mentre l'applicazione insiemistica per la φ riguardi argomenti che sono n-uple, quella combinatoria per la X riguarda un singolo argomento (cioè x_0). Tuttavia tale applicazione viene iterata (su $x_1 \dots x_{n-1}$).*

Questo non dipende dal fatto che nel calcolo dei combinatori non si possano rappresentare delle n-uple; essa è solo una scelta di semplicità "suffi ciente" per i nostri scopi (così come quella della riduzione debole). Inoltre nei lemmi per la rappresentazione delle funzioni parziali ricorsive si devono comunque introdurre delle rappresentazioni (di coppie) e queste farebbero confusione con quelle per gli argomenti.

Teorema. 4.1 (di rappresentazione) *Ogni funzione parziale ricorsiva può essere rappresentatan in senso debole da qualche termine combinatorio.*

Di questo enunciato non daremo la dimostrazione completa, ma solo dei cenni ad essa ed ai lemmi che usa.

Dim. 4.2 (cenno) *Per il teorema della forma normale delle funzioni parziali ricorsive* $\varphi(x_0,\ldots,x_{n-1}) = f(\mu_y(g(y,x_0,\ldots,x_{n-1}) = 0))$*, data una tale* φ *esistono due funzioni primitive ricorsive* f *e* g *che la definiscono mediante* μ*. Pertanto basta dimostrare che ogni funzione primitiva ricorsiva è rappresentabile combinatoriamente e che date una* g *ed una* f *così rappresentabili, è pure rappresentabile la minimizzazione della* g *e la composizione di una funzione rappresentabile con la* f*.*

Noi trascureremo la rappresentabilità della μ *e accenneremo a quella delle funzioni ricorsive primitive (che comporta anche un caso di composizione).*

Lemma. 4.2 *Ogni funzione primitiva ricorsiva è rappresentabile combinatoriamente in senso debole.*

Dim. 4.3 (cenno) *Occorre dimostrare che le funzioni di base per la ricorsione primitiva sono rappresentabili. Queste sono:*

- 0*, inteso come una funzione (costante) di zero argomenti (o anche come una funzione costante di un argomento);*
- s*, successore;*
- U_i^n*, proiezioni.*

Esse sono immediatamente rappresentabili poichè:

- $\overline{0}$ *cioè* KI*, è già stato rappresentato (c'è una piccola modifica per lo* 0 *con un argomento);*
- $\overline{s}$ *altro non è che SB poichè* $SB\overline{n} = \overline{n+1} = \overline{s(n)}$*;*
- $\overline{U_i^n}$ *si ottengono dagli* $X \equiv [x_0,\ldots,x_{n-1}]x_i$*, poichè dal teorema di riduzione si ha che* $X\overline{x}_0\,\overline{x}_1\cdots\overline{x}_{n-1} = \overline{x}_i$*, ove gli* x_i *siano numeri naturali.*

Una volta ottenuta la rappresentabilità di tali funzioni di base occorre dimostrare che la rappresentabilità si conserva per composizione (multipla) e ricorsione primitiva.

Per la composizione si osservi che date f *ad* m *argomenti e* $g_0,\ldots,g_{m-1}$ *tutte ad* n *argomenti, il rappresentanto della* h *tale che:*

$$h(x_0,\ldots,x_{n-1}) = f(g_0(x_0,\ldots,x_{n-1}),\ldots,g_{m-1}(x_0,\ldots,x_{n-1});$$

può ovunque essere definito dai rappresentanti $\overline{f}$*,* $\overline{g}_0$*, ... ,* $\overline{g}_{n-1}$ *mediante:*

$$\overline{h} = [x_0,\ldots,x_{n-1}]\overline{f}(\overline{g}_0x_0x_1\cdots x_{n-1})\cdots(\overline{g}_{m-1}x_0x_1\cdots x_{n-1}).$$

Che tale termine combinatorio $\overline{h}$ *sia effettivamente il rappresentante richiesto segue ancora dal teorema di riduzione multipla:*

$$\overline{h}\,\overline{x}_1\cdots\overline{x}_{n-1} = \overline{f}(\overline{g}_0\,\overline{x}_0\cdots\overline{x}_{n-1})\cdots(\overline{g}_{m-1}\,\overline{x}_0\cdots\overline{x}_{n-1}).$$

Per la ricorsione primitiva si procede in due passi:

I) *Si dimostra che la rappresentabilità si conserva nel caso che la funzione g, nella clausola di base della ricorsione, sia una funzione a zero argomenti.*

II) *Si estende tale dimostrazione al caso di più argomenti.*

Noi omettiamo il passo II) *ed accenniamo al primo.*

Nel caso di zero argomenti definiamo una φ ad un argomento mediante una ricorsione primitiva semplificata del tipo:

$$\begin{cases} \varphi(0) = c; & \textit{(costante)} \\ \varphi(n+1) = \chi\big(n, \varphi(n)\big). \end{cases}$$

L'idea su cui si basa la dimostrazione è la seguente. Consideriamo la φ (che è funzione insiemistica) come l'insieme delle coppie:

$$\Big\{\langle 0, c\rangle, \langle 1, \varphi(n)\rangle, \ldots, \langle n, \varphi(n)\rangle, \ldots\Big\}.$$

In tali coppie si ha che $\varphi(n+1)$ è uguale a $\chi\big(n, \varphi(n)\big)$; per esempio si ha la coppia $\langle 1, \chi(0, c)\rangle$.

Pertanto possiamo enumerare tale insieme mediante una funzione f dipendente da χ, $f = q_\chi$, tale che:

$$f\big(n, \varphi(n)\big) = \Big\langle n+1,\ \varphi(n+1)\Big\rangle = \Big\langle n+1,\ \chi\big(n, \varphi(n)\big)\Big\rangle.$$

Se saremo in grado di rappresentare tale f mediante un termine combinatorio F potremo ottenere la rappresentazione della generica coppia $\langle n+1,\ \varphi(n+1)\rangle$ mediante iterazione fatto sulla F. In altre parole potremo potremo ridurre il nostro caso semplificato (zero argomenti) di ricorsione primitiva a una semplice iterazione usando F^n, cioè utilizzando gli iteratori come nel lemma d'iterazione.

Per precisare tale idea occorre tuttavia essere in grado di rappresentare coppie di numeri naturali e di manipolarle in modo da poter estrarre dal rappresentante di $\langle n,\ \varphi(n)\rangle$ il secondo componente $\varphi(n)$ una volta individuato il primo.

4.1 Rappresentazione delle coppie insiemistiche

Una prima idea per rappresentare una coppia di termini combinatori con un'altro termine è quella di considerare l'applicazione di due termini componenti. Tuttavia ciò non funziona nonostante che l'applicazione goda (per definizione) di alcune proprietà delle coppie insiemistiche. Esistono funzioni di "decomposizione" p ed s tali che:

$$p\big(\langle A, B\rangle\big) = A \quad \text{e} \quad s\big(\langle A, B\rangle\big) = B.$$

Ciò non è possibile con l'applicazione poichè nella rappresentazione l'uguale insiemitico deve diventare l'uguale combinatorio (definito da un processo di

calcolo). Infatti si ha ad esempio che non può esistere un P che si comporti come il p insiemistico, cioè non esiste un termine combinatorio P tale che:

$$P(XY) = X, \quad \text{per ogni termine combinatorio } X \text{ e } Y.$$

Altra idea. Definiamo $D' \equiv [x, y, z]zxy$, dati X e Y abbiamo così il termine combinatorio $D'XYZ = ZXY$ per ogni Z. Se prendiamo $Z \equiv \mathbf{K}$

$$D'XYZ = \mathbf{K}XY = X;$$

pertanto possiamo definire come rappresentante della coppia X e Y il termine combinatorio $D'XY$ e come P:

$$P \equiv [x]xK, \quad (\text{di modo che } P(D'XY) = D'XY\mathbf{K} = X\ ).$$

Oltre a tale D' esistono parecchi combinatori che consentono di rappresentare le coppie insiemistiche. Il combinatore di "coppia" che chiameremo *diade*, che serve nel nostro caso, usa come termini P ed S (per rappresentare s) dei termini derivati da rappresentanti numerici. Infatti definiamo:

$$D \equiv [x, y, z]z(\mathbf{K}y)x.$$

Si ha allora che $DXY\overline{0}$ per il teorema di riduzione:

$$DXY\overline{0} = \overline{0}(\mathbf{K}Y)X = \mathbf{KI}(\mathbf{K}Y)X = \mathbf{I}X = X;$$

$$DXY\overline{n+1} \ = \ \cdots \ = \ Y.$$

Cioè da DXY possiamo estrarre X o Y mediante P ed S come i precedenti, ma definiti a partire dai rappresentanti numerici.

Usando tale D possiamo dimostrare il seguente lemma.

Lemma. 4.3 *Esiste un combinatore* $\mathbf{R}$*, che ha forma normale, tale che:*

1) $\mathbf{R}XY\overline{0} \ = \ X$*;*

2) $\mathbf{R}XY\overline{n+1} = Y\overline{n}(\mathbf{R}XY\overline{n})$*.*

Tale operatore di ricorsione può essere usato allora per rappresentare la ricorsione primitiva semplificata ($n = 0$) *poichè* X *può essere preso come il rappresentante di* c ($\overline{c}$) *ed* Y *come il rappresentante di* χ ($\overline{\chi}$)*, per cui, dalla 1) e 2)* $\mathbf{R}XY$ *rappresenta* φ *in accordo a* $\varphi(0) = c$ *e* $\varphi(n+1) = \chi\big(n, \varphi(n)\big)$*.*

Dim. 4.4 (cenno) *Si segue l'idea precedente di numerare i rappresentanti delle coppie* $\langle n, \varphi(n)\rangle$ *mediante il rappresentante di* $f = q_\chi$*. Pertanto si definisce un termine* Q *che rappresenti* q*, cioè tale che* QY *rapresenti la* f*, quindi la* Y *rappresenta la* χ*. Tale* Q *è costruita tramite la diade* D*, infatti si ha*

$$Q \equiv [x, d]D\big(SB(d\overline{0})\big)\big(x(d\overline{0})(d\overline{1})\big);$$

ove la x *servirà per* χ *e la* d *per* $\langle n, \varphi(n)\rangle$*. Mediante tale* Q *si può poi costruire il termine combinatorio* $\mathbf{R}$ *tramite iterazione ed utilizzando l'estrazione dei componenti della coppia rappresentata ottenendo la 1) e la 2). In definitiva dal lemma di iterazione si ottiene un lemma di ricorsione primitiva semplificata.*

Lemma. 4.4 *Qualsiasi funzione rappresentabile combinatoriamente è parziale ricorsiva.*

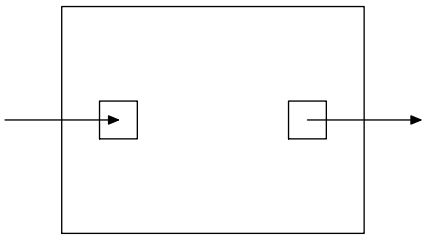
Dim. 4.5 *No.*

Non si fa perchè è ovvio a causa dell'equivalenza fra calcolabilità secondo Turing e funzioni parziali ricorsive.

Capitolo 5

Indecidibilità

La teoria e "filosofia" di A. Turing ha chiarito che "tutto il fattibile" è tale se e solo se qualche macchina di Turing lo può fare.



Se si danno in entrata dei dati e si aspettano in uscita i risultati, allora non si può distinguere se all'interno della scatola vi sia un "uomo" o un'opportuna macchina di Turing (M. di T.). Turing si è inoltre occupato delle limitazioni di tale "fattibilità totale"[1]. La prima di tali limitazioni è data dal teorema dell'arresto: non può esistere alcuna M. di T. che, ricevendo in entrata la descrizione di un'altra M. di T. qualsiasi e dei relativi dati d'ingresso, sia in grado di dare in uscita per il si e per il no risposta al quesito "il calcolo della seconda macchina da un risultato, cioè la seconda macchina si arresta?".

Dal teorema dell'arresto derivano numerosi altri risultati di *indecidibilità*. Per esempio:

1. Non è possibile decidere sempre se una funzione (calcolabile) sia definita per certi argomenti;

2. Non è possibile decidere sempre se una funzione (calcolabile) abbia per qualche argomento un certo valore;

[1] Precedentemente a Turing, K. Gödel aveva trovato dei risultati a proposito delle limitzioni della dimostrabilità totale in matematica.

3. Come corollario del 2) si ha che non è possibile calcolare una funzione reale di variabili reali discontinua[2];

4. Non è possibile decidere sempre se due funzioni (calcolabili) siano uguali oppure no.

5. Si recuperano i risultati di indecidibilità della logica matematica;

⋮

Buona parte di tali risultati di indecidibilità, provenienti dal teorema dell'arresto sono "condensati" in un (pesante) teorema (di Rice-Shopiro-Myhill) che asserisce qualcosa di grossolanamente simile a "qualsiasi cosa che sia sistematicamente prevedibile a proposito delle funzioni effettivamente calcolabili è banale".

I risultati di indecidibilità servono nello sviluppo del software. Per esempio, se si vuole progettare un linguaggio di programmazione che abbia la garanzia di dare programmi che diano sempre un risultato (evitando ad esempio cicli non voluti dal programmatore) allora tale linguaggio *non* può essere universale, cioè non deve consentire che sia programmabile qualsiasi funzione calcolabile. Tali limitazioni dell'universalità possono essere suggeriti dalla teoria dell'indecidibilità.

5.1 Indecidibilità per i termini combinatori

Mentre l'indecidibilità secondo Turing riguarda inizialmente solo l'agente di calcolo (M. di T.), per il calcolo combinatorio si hanno subito dei risultati d'indecidibilità che riguardano i programmi idealizzati espressi dai termini combinatori. In particolare avremo un risultato simile al 4. precedente, che spesso sostituisce (tramite dei corollari immediati) il teorema di Rice-Shopiro-Myhill.

Si parte con la "Gödellizzazione" dei termini combinatori. Ciò significa che ogni termine combinatorio X viene identificato da un numero naturale $gd(X)$ secondo una enumerazione, data da una funzione gd che abbia delle caratteristiche di fattibilità. In particolare si richiede che esistano una funzione ricorsiva (totale) μ a due argomenti ed una ricorsiva totale ν ad un argomento tali che:

$$\mu\big(gd(X), gd(Y)\big) = gd(XY) \quad \forall \text{ termine combinatorio } X \text{ e } Y;$$

$$\nu(n) = gd(\overline{n}) \quad \forall\, n \in \mathbb{N}.$$

Noi trascureremo come definire una tale gd, poichè non dimostreremo il teorema.

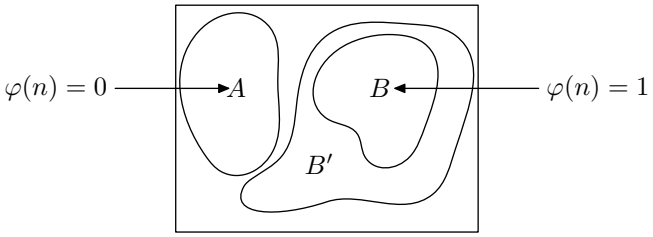
[2]La calcolabilità riferita a funzioni tra numeri naturali si estende a quelle fra reali considerando ad esempio le rappresentazioni posizionali dei numeri reali che sono assegnabili come funzioni fra numeri naturali

Def. 5.1 *Diremo che due insiemi $\mathcal{A}$ e $\mathcal{B}$ di numeri naturali sono* ricorsivamente separabili *quando esiste una funzione ricorsiva (totale) con valori* 0 *ed* 1 *tale che:*

- *se* $n \in \mathcal{A} \implies \varphi(n) = 0$*;*
- *se* $n \in \mathcal{B} \implies \varphi(n) = 1$*.*

Si noti che il semplice fatto che φ sia funzione implica che $\mathcal{A} \cap \mathcal{B} = \emptyset$. Tuttavia non vale il viceversa, poichè richiediamo che φ sia anche ricorsiva.

In sostanza la φ è una funzione calcolabile che sia la funzione caratteristica di un insieme $\mathcal{B}'$ contenente $\mathcal{B}$ e disgiunto da $\mathcal{A}$.



Def. 5.2 *Diremo che un insieme $\mathcal{A}$ è* ricorsivo *quando la sua funzione caratteristica φ:*

$$\varphi(n) = \begin{cases} 1 & \text{se } n \in \mathcal{A}; \\ 0 & \text{altrimenti;} \end{cases}$$

è ricorsiva.

Similmente per una relazione fra numeri naturali diremo che è ricorsiva *se la sua funzione caratteristica lo è:*

$$\varphi(n, m) = \begin{cases} 1 & \text{se } \langle n, m \rangle \in r; \\ 0 & \text{altrimenti.} \end{cases}$$

Tali definizioni relative a numeri naturali sono estese ai termini combinatori considerando i numeri di Gödel che li identificano.

Def. 5.3 *Diremo che un insieme A di termini combinatori è* chiuso rispetto all'uguaglianza (debole) *quando:*

$$X \in A \text{ e } Y = X \text{ implica } Y \in A.$$

Teorema. 5.1 (inseparabilità ricorsiva) *Nessuna coppia di insiemi A e B non vuoti di termini combinatori chiusi rispetto all'uguaglianza debole è ricorsivamente separabile.*

Dim. 5.1 *No.*

Contrariamente all'uso precedente di nozioni insiemistiche (che era evitabile), ora le nozioni di insieme sono essenziali. In effetti le nostre definizioni e teoremi non sono propriamente della teoria dei combinatori.

Corollario 5.1 (A) *Nessun insieme non banale (non vuoto o non totale) di termini combinatori che sia chiuso rispetto all'uguaglianza (debole) può essere ricorsivo.*

Dim. 5.2 *Basta prendere, nel teorema d'inseparabilità ricorsiva, la coppia d'insiemi formata da tale insieme e dal suo complemento (rispetto all'isieme di tutti i termini combinatori).*

Corollario 5.2 (B) *L'insieme di tutti i termini combinatori aventi forma normale non è ricorsivo.*

Dim. 5.3 *Basta vedere che non è banale ed usare il corollario A (non è vuoto perchè gli atomi stanno in esso e non è totale poichè vi sono termini combinatori senza forma normale).*

Si noti che questo corollario corrisponde al teorema dell'arresto.

Corollario 5.3 (C) *La relazione di uguaglianza debole non è ricorsiva.*

Dim. 5.4 *Basta considerare per assurdo l'insieme dei termini uguali a un dato termine combinatorio. Allora dalla funzione caratteristica dell'uguaglianza debole possiamo costruire quella di tale insieme in modo che resti ricorsiva.*

Capitolo 6

Introduzione al λ-calcolo

Per introdurre il λ-calcolo l'idea di partenza è quella di effettuare un'inversione logica del teorema di riduzione dei termini combinatori:

$$\big([x]M\big)N \ \rhd \ [N/x]M;$$

da questa si potevano trovare le costanti riducibili.

Pertanto se incorporiamo nelle nostre definizioni tale asserto non avremo bisogno delle costanti riducibili **I**, **K**, **S**. Il primo passo sarà quello di incorporare l'astrazione combinatoria nei termini.

6.1 λ-termini

Definiamo delle sequenze di variabili e di costanti per definire gli atomi, ma non richiediemo che fra le costanti ci siano **I**, **K**, **S**. Pertanto gli atomi potrebbero anche provenire solo dalla sequenza infinita delle variabili.

Definiamo i λ-termini con le clausole:

a - Ogni atomo è un λ-termine;

b - Se X e Y sono λ-termini allora XY è un λ-termine;

c - Se x è una variabile e X è un λ-termine, allora $\lambda x.X$ è un λ-termine;

($\lambda x.X$ è da pensare come $[x]X$).

Nella clausola b si definisce l'applicazione (come nel caso dei termini combinatori), nella clausola c si definisce invece la λ-astrazione che non è un operatore esterno sui termini, ma è incorporato in essi. Tale λ-astrazione è definita analogamente alla applicazione, cioè:

- $\lambda x'.X' \equiv \lambda x''.X''$ implica $x' \equiv x''$ e $X' \equiv X''$;

- Solite clausole di disgiunzione[1].

Notazioni semplificative: - Parentesi sintattiche omesse se associate a sinistra (come per i termini combinatori);

- Parentesi sintattiche per λ-astrazione omesse se associate a destra.

Esempio. 17

$$X(((\lambda x.Y)X)Y) \textit{ può diventare } X((\lambda x.Y)XY).$$

$$\lambda x.(\lambda y.(\lambda z.X)) \textit{ può diventare } \lambda xyz.X \quad \textit{(come per } [x,y,z]X\textit{).}$$

Per visualizzare la struttura di un λ-termine ora non bastano gli alberi binari (con sola etichettatura delle foglie). Ci serviremo di alberi "al più binari" con i nodi unari corrispondenti alle λ-astrazioni etichettati dalla variabile di astrazione.

Esempio. 18 *Alcune rappresentazioni grafiche.*

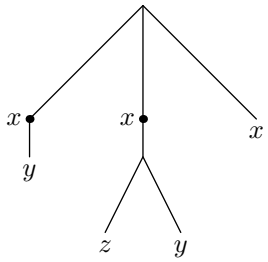


Figura 6.1: $\big((\lambda x.y)(\lambda x.zy)\big)x$

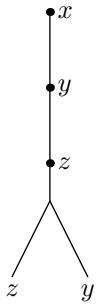


Figura 6.2: $\lambda xyz.zy = \lambda x.\big(\lambda y.(\lambda z.zy)\big)$

[1] Atomi, λ-termini composti per applicazione, λ-termini composti per astrazione formano "insiemi" disgiunti.

Def. 6.1 *Diremo che* X occorre *in* Y *e scriveremo* $X \in Y$ *quando ciò può essere dedotto da:*

- $X \in X$;
- *Se* $X \in U$ *o* $X \in V$ *allora* $X \in UV$;
- *Se* $X \in V$ *allora per ogni* x *variabile* $X \in \lambda x.V$.

È da notare che $x \notin \lambda x.y$ *per ogni* $y \neq x$.

Siccome $\lambda x.$ è da pensare come "$[x]$" e sappiamo che nell'astrazione combinatoria i nomi delle variabili d'astrazione non contano ($[x]X = [y]([y/x]X)$), allora dobbiamo distinguere le occorrenze di variabili a seconda che "contino" o no.

Def. 6.2 *Un'occorrenza di una variabile* x *in un* λ*-termine* Y *si dirà* vincolata *—cioè ivi il nome di* x *non conta— quando tale occorrenza sia presente in un sottotermine di* Y*, che sia del tipo* $\lambda x.X$ *(cioè stia in* X*). Se ogni occorrenza di* x *in* Y *è vincolata diremo che* x *è* vincolata *in* Y*. Se invece esiste almeno una occorrenza di* x *in* Y *che non sia vincolata diremo che* x *è* libera *in* Y.

Con le nostre convenzioni di rappresentazione grafica tali nozioni sono più semplici: In particolare per $X \equiv x$ avremo una foglia.

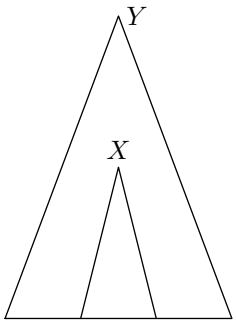
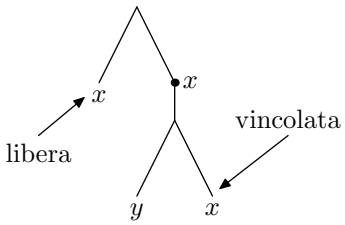


Figura 6.3: $X \in Y$ quando l'albero di X è un sottoalbero di Y.

Sempre considerando la rappresentazione grafica, possiamo notare che x occorre libera in Y quando da almeno una foglia etichettata con x si può risalire alla radice senza incontrare un nodo (unario) etichettato con x. Un'occorrenza di x in Y è vincolata se risalendo da tale foglia si incontra almeno un tale nodo.



Def. 6.3 *Si definisce la* sostituzione $[N/x]M$ del λ*-termine* N al posto *della variabile* x nel λ*-termine* M *con le clausole*[2] *del tipo:*

- $[N/x]x \equiv N$;
- $[N/x]a \equiv a$ *per* $a \not\equiv x$ *atomo;*
- $[N/x]UV \equiv \big([N/x]U\big)\big([N/x]V\big)$;
- $[N/x]\lambda x.X \equiv \lambda x.X$ *(poichè le occorrenze di* x *sono vincolate);*
- $[N/x]\lambda y.X \equiv \begin{cases} \text{si distinguono due casi a seconda che } y \text{ occorra in } N \\ \text{(libera) in tal caso si rinomina } y. \end{cases}$

Def. 6.4 *Diremo che un* λ*-termine* X *è congruente ad un* λ*-termine* Y *e d'ora in poi scriveremo* $X \equiv Y$ *(non più identità) quando si può ottenere* Y *da* X *mediante una "ridenominazione completa" di variabili in occorrenze vincolate.*

Esempio. 19 $\lambda x.yx \equiv \lambda z.yz$

Coerentemente alla nostra interpretazione della λ-astrazione le differenze tra λ-termini congruenti saranno da ignorare. In sostanza il nostro "$\equiv$" (che non è un identità) sarà da pensare equivalente all'identità "$\equiv$" per i termini combinatori.

Def. 6.5 *Un* λ*-termine della forma* $(\lambda x.M)N$ *che occorra in* X *si dirà un* redex *di* X*. Diremo sua contrazione il* λ*-termine* $[N/x]M$*. Diremo che* X *si riduce a* Y *e scriveremo* $X \rhd Y$ *(nella letteratura ci sono altre definizioni) quando* Y *si ottenga da* X *tramite una sequenza finita di contrazioni. Formalmente quando ciò provenga da:*

$$(\lambda x.M)N \rhd [N/x]M;$$

$$X \rhd X;$$

mediante le solite regole di deduzione.

Mediante simmetrizzazione e congruenza si definisce l'uguaglianza debole "$=$".

Per λ-β riduzione s'intende la relazione ottenuta aggiungendo all'assioma di contrazione precedente quello di cambiamento delle variabili vincolate. Continueremo ad indicarla con "$\rhd$", ma la useremo solo nei teoremi, mentre negli esempi od esercizi useremo la riduzione senza cambiamenti di variabili vincolate.

L'uguaglianza debole per λ-termine viene anche chiamata λ-conversione o λ-uguaglianza.

Esempio. 20 *Vediamo come esercizio alcune applicazioni di riduzioni partendo da redex.*

[2] L'ultima clausola non è stata formalizzata poiché rappresenta un "caso" abbastanza complicato.

$$\begin{array}{rcl} (\lambda x.xy)F & \rhd & [F/x]xy \\ & \equiv & Fy. \end{array} \tag{6.1}$$

$$\begin{array}{rcl} (\lambda x.y)F & \rhd & [F/x]y \\ & \equiv & y. \end{array} \tag{6.2}$$

$$\begin{array}{rcl} \underbrace{\big(\lambda x.(\underbrace{\lambda y.yx}_{})z\big)v}_{redex} & \rhd & \big(\lambda x.[z/y]yx\big)v \\ & \equiv & (\lambda x.zx)v \\ & \rhd & [v/x]zx \\ & \equiv & zv. \end{array} \tag{6.3}$$

Nel caso di cui sopra abbiamo scelto di contrarre per primo il redex interno, vediamo cosa succede se avessimo scelto il redex più esterno:

$$\begin{array}{rcl} \underbrace{\big(\lambda x.(\underbrace{\lambda y.yx}_{})z\big)v}_{redex} & \rhd & [v/x](\lambda y.yx)z \\ & \equiv & (\lambda y.yv)z \\ & \rhd & [z/y]yv \\ & \equiv & zv. \end{array} \tag{6.4}$$

Analogamente a quanto detto per i termini combinatori, anche nel λ-calcolo, pur seguendo strade diverse otteniamo il medesimo risultato.

$$\begin{array}{rcl} (\lambda x.xxy)(\lambda x.xxy) & \rhd & [\lambda x.xxy/x]xxy \\ & \equiv & (\lambda x.xxy)(\lambda x.xxy)y \\ & \rhd & [(\lambda x.xxy)/x]xxyy \\ & \equiv & (\lambda x.xxy)(\lambda x.xxy)yy \\ & \vdots & \vdots \\ & \equiv & (\lambda x.xxy)(\lambda x.xxy)y\cdots y. \end{array} \tag{6.5}$$

La riduzione può continuare con contrazioni infinite.

$$\begin{array}{rcl} (\lambda x.xx)(\lambda x.xx) & \rhd & [\lambda x.xx/x]xx \\ & \equiv & (\lambda x.xx)(\lambda x.xx) \\ & \rhd & [(\lambda x.xx)/x]xx \\ & \equiv & (\lambda x.xx)(\lambda x.xx) \\ & \vdots & \vdots \\ & \equiv & (\lambda x.xx)(\lambda x.xx). \end{array} \tag{6.6}$$

La riduzione può continuare con contrazioni infinite (effettive), ma di fatto "non contraggono".

Da tali esempi si vede come si definisce per il λ-termine la forma normale o la sua assenza come in 6.5 e 6.6. L'equazione 6.3 ci preannuncia il primo teorema di

Church-Rosser per i λ-termini (che non dimostreremo) così come il corollario sull'unicità della forma normale, che vale a meno di congruenze e che ha la stessa dimostrazione vista. Vale pure il secondo teorema di Church-Rosser (con la medesima dimostrazione) sempre a meno di congruenze e relativi corollari.

Come per i termini combinatori non vale la regola di estensionalità.

Esempio. 21 *Non estensionalità.*

$$X \equiv \lambda x.yx,\ Y \equiv y \quad \textit{allora per ogni } \lambda\textit{-termine } V,\ XV = yV \equiv YV.$$

Tuttavia $X \neq Y$ poichè sono in forma normale e non congruenti.

6.2 Alberi di De Bruijn

Gli Alberi di De Bruijn costituiscono una definizione alternativa dei λ-termini che evita il problema delle variabili vincolate. Quindi non c'è bisogno di introdurre congruenze, la riduzione debole diventa la λ-β riduzione. Oltre a tali vantaggi per la teoria essi possono servire nell'implementazione del linguaggio LISP che accenneremo, poichè permettono controlli di congruenza immediati.

Ritorniamo alla rappresentazione grafica di un λ-termine

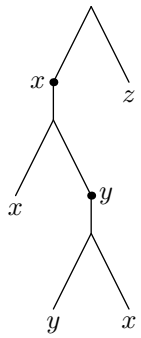


Figura 6.4: $\big(\lambda x.x(\lambda y.yx)\big)z$

e consideriamo il ruolo delle variabili.

- La z è una variabile libera che possiamo considerare come la prima nella successione delle variabili libere. Quindi la possiamo chiamare 1.
- La x è invece vincolata (come la y). Il suo ruolo nell'albero è identificato dal nodo unario etichettato x. Se vogliamo ignorare le congruenze , non importa che sia stata chiamata x, a tale fine non è necessario avere tale nodo, basta sapere dov'è. Tale posizione è univocamente determinata dal numero di nodi binari che dobbiamo risalire dalla foglia nominati x per raggiungere tale nodo unario, cioè dal *livello d'astrazione*. Nel nostro caso è 1 o 2. Possiamo allora convenire che tale occorrenza di x nella foglia sia indicata dall'opposto -1 di tale livello d'astrazione.

- Per le y valgono ancora le osservazioni precedenti e continuiamo a chiamarla -1.

Pertanto il nostro albero al piú binario diventa un puro albero binario con foglie etichettate da numeri interi.

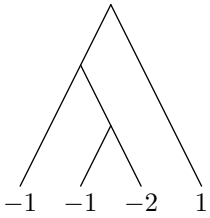


Figura 6.5: Albero di De Bruijn.

Si noti che per le variabili libere possiamo stabilire una corrispondenza biunivoca fra i loro nomi e l'intero positivo che la identifica. Ciò non vale per le variabili vincolate. Nel nostro esempio il nome x corrisponde sia a -1 che a -2, invece il numero -1 corrisponde sia al nome x che al nome y.

Anche se la dimostrazione è un po' complicata è chiaro che da ogni albero al più binario di un λ-termine corrisponde un unico albero del secondo tipo che chiameremo *albero di De Bruijn*. Viceversa dato un albero binario con foglie etichettate da interi possiamo sempre costruire almeno un albero del primo tipo con opportune altre convenzioni Si può dimostrare che tutti gli alberi così costruiti sono quelli di λ-termini congruenti.

Esempio. 22 *Vediamo alcune trasformazioni.*

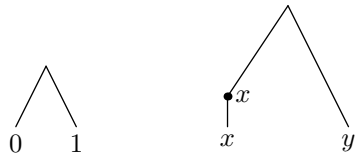


Figura 6.6: $(\lambda x.x)y$

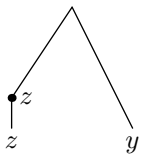
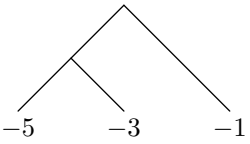
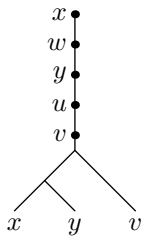


Figura 6.7: altra possibile interpretazione, $(\lambda z.z)y$.

Similmente, convenendo che in mancanza di nodi binari si conteggino nodi d'astrazione, si ha che:



ci da ad esempio λxwyuv.xyv:



In conclusione possiamo ridefinire i λ-termini a meno di congruenze, semplicemente come alberi binari su foglie di interi qualsiasi. L'uso delle variabili vincolate è solo per facilità di comprensione.

6.3 Uguaglianza estensionale

Sinora l'uguaglianza per i termini combinatori e i λ-termini era debole poiché non valeva la regola di estensionalità 2.6. Tale uguaglianza è sufficiente per i risultati visti (teorema di Church-Rosser e conseguenze, esempi di "consistenza logica" ed altri), tuttavia non è sufficiente per dimostrare l'equivalenza tra λ-calcolo e calcolo dei combinatori, come vedremo.

Si noti che possiamo considerare i λ-termini ed i termini combinatori come programmi idealizzati per un linguaggio di programmazione senza o con primitive[3].

Con "=" identifichiamo così, in ambedue i casi, un'uguaglianza di programmi idealizzati comprensivi dei loro "dati". Tale equivalenza asserisce che due programmi o hanno lo stesso risultato (forma normale eventualmente a meno di congruenze) oppure il loro calcolo può essere ricondotto sempre allo stesso punto nel senso del secondo teorema di Church-Rosser.

Invece un'uguaglianza che soddisfi la regola (est) riguarda programmi non necessariamente completi dei loro dati (in est il dato è V). Quindi mentre in Teoria degli Insiemi si ha una sola nozione di uguaglianza, proveniente dall'assioma di estensionalità, l'uguaglianza dal punto di vista operativo o *intensionale* si divide in due diverse nozioni.

Consideriamo ora tre regole diverse da (est).

[3]Nel caso di λ-termini in luogo di primitive abbiamo la λ-astrazione che può essere considerata come la generazione di una procedura con passaggio di parametri secondo la nostra definizione di contrazione $(\lambda x.M)N \rhd [N/x]M$.

(ζ) $Xx = Yx$ per ogni variabile x che non occorra in X o Y, implica $X = Y$.

È una semplificazione della (est), ma come quest'ultima ha una premessa infinita poiché le variabili che non occorrono nei due termini sono infinite, tuttavia si può considerare come una singola premessa. Tale regola può riguardare sia λ-termini che termini combinatori.

(ξ) $X = Y$ implica $[x]X = [x]Y$ per ogni variabile x (che riguarda solo i termini combinatori).

A differenza della (est) e (ζ) la premessa è singola. Infine per i λ-termini abbiamo:

(η) $\lambda x.Xx = X$ per ogni variabile x che non occorra in X.

Si noti che (η) non è un implicazione e puó essere essere aggiunta come assioma ad una definizione di uguaglianza. In pratica essa amplia il tipo dei redex da considerare. Essa corrisponde alla terza clausola della definizione di astrazione per i termini combinatori.

Teorema. 6.1 *La regola (est) è equivalente a: (ζ) nel caso dei λ-termini o termini combinatori, (ξ) nel caso dei termini combinatori ed (η) nel caso dei λ-termini.*

Dim. 6.1 *No.*

Nell'ultimo caso l'uguaglianza ottenuta (fra λ-termini) viene detta $\lambda\beta\eta$ *convertibilità*.

6.4 Riduzione forte (cenno)

Nel caso dell'uguaglianza debole potevamo caratterizzarla come la chiusura simmetrica della riduzione "$\triangleright$" (debole per i termini combinatori o $\lambda\beta$ per i λ-termini). Ci si può chiedere se esista un'analoga relazione "$\succ$" per il caso dell'uguaglianza estensionale. La risposta è affermativa e la "$\succ$" viene detta riduzione forte. Pertanto si può pensare all'uguaglianza forte come ad un'equivalenza generata da un "processo di calcolo". Tuttavia tale processo è più complicato delle varie riduzioni.

In corrispondenza alle uguaglianze forti si hanno le varianti delle nozioni relative alla forma normale dei teoremi di Church-Rosser e d'inseparabilità ricorsiva sia per i λ-termini che per i termini combinatori. Con alcune limitazioni si avranno anche delle varianti dei risultati di decidibilità che vedremo in seguito.

Ciò permette di utilizzare le uguaglianze estensionali per risolvere problemi di decidibilità che nascono in Teoria degli Insiemi a patto di poterli tradurre in problemi relativi a λ-termini e termini combinatori[4].

[4]Tale traduzione può essere assai difficile.

6.5 Equivalenza tra λ-calcolo e calcolo dei combinatori

In questa parte del testo verificheremo l'equivalenza tra λ-calcolo e calcolo dei combinatori, per facilitare la notazione il simbolo "=" indicherà solo l'uguaglianza estensionale.

Def. 6.6 *Dato un λ-termine X diremo suo* H-trasformato *il termine combinatorio X_H, ottenuto per induzione combinatoria, tale che:*

- $a_H \equiv a$ *per ogni atomo*[5]*;*

- $(UV)_H \equiv U_H V_H$*;*

- $(\lambda x.X)_H \equiv [x]X_H$*.*

Def. 6.7 *Dato un termine combinatorio X diremo suo λ-trasformato il λ-termine X_λ tale che:*

- $a_\lambda \equiv a$ *per ogni atomo diverso dalle costanti riducibili;*

- $\mathbf{I}_\lambda \equiv \lambda x.x$*;*

- $\mathbf{K}_\lambda \equiv \lambda xy.x$*;*

- $\mathbf{S}_\lambda \equiv \lambda xyz.xz(yz)$*;*

- $(UV)_\lambda \equiv U_\lambda V_\lambda$*.*

Esempio. 23 *Vediamo alcune H e λ trasformazioni.*

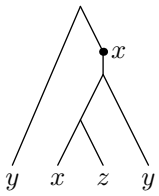
$$\begin{aligned}\big(y(\lambda x.xzy)\big)_H &\equiv y_H(\lambda x.xzy)_H\\ &\equiv y\big([x](xzy_H)\big)\\ &\equiv y\big([x]xzy\big)\\ &\equiv y\big(\mathbf{S}([x]xz)(\mathbf{K}y)\big)\\ &\equiv y\Big(\mathbf{S}\big(\mathbf{SI}(\mathbf{K}z)\big)(\mathbf{K}y)\Big).\end{aligned}$$

[5]Si suppone che tutti gli atomi siano gli stessi nei due sistemi (λ-calcolo e calcolo dei combinatori), salvo le costanti riducibili.

$$\Big(y\Big(\mathbf{S}\big(\mathbf{SI}(\mathbf{K}z)\big)(\mathbf{K}y)\Big)\Big)_\lambda$$
$$\equiv$$
$$y_\lambda\Big(\mathbf{S}\big(\mathbf{SI}(\mathbf{K}z)\big)(\mathbf{K}y)\Big)_\lambda$$
$$\equiv$$
$$y\Big(\Big(\mathbf{S}\big(\mathbf{SI}(\mathbf{K}z)\big)\Big)_\lambda(\mathbf{K}y)_\lambda\Big)$$
$$\equiv$$
$$y\Big(\Big(\mathbf{S}_\lambda\big(\mathbf{SI}(\mathbf{K}z)\big)_\lambda\Big)(\mathbf{K}_\lambda y_\lambda)\Big)$$
$$\equiv$$
$$y\Big(\Big((\lambda xyz.xz(yz))\big((\mathbf{SI})_\lambda(\mathbf{K}z)_\lambda\big)\Big)\big((\lambda xy.x)y\big)\Big)$$
$$\equiv$$
$$y\Big(\Big((\lambda xyz.xz(yz))\big((\mathbf{S}_\lambda\mathbf{I}_\lambda)(\mathbf{K}_\lambda z_\lambda)\big)\Big)\big((\lambda xy.x)y\big)\Big)$$
$$\equiv$$
$$y\Big(\Big((\lambda xyz.xz(yz))\Big(\big((\lambda xyz.xz(yz))(\lambda x.x)\big)\big((\lambda xy.x)z\big)\Big)\big((\lambda xy.x)y\big)\Big).$$

Pur essendo partiti dall'H-trasformazione di $\big(y(\lambda x.xzy)\big)$ non *abbiamo riottenuto lo stesso termine applicando la* λ*-trasformazione al risultato intermedio. Potrebbe valere l'uguaglianza debole?*

Per verificarne l'uguaglianza (debole) dovremmo cercare la forma normale del risultato della seconda equazione, il calcolo è lungo, tuttavia possiamo considerare un'occorrenza di tale risutato: $(\lambda xy.x)y$. Per riduzione e cambiamento di variabile si ottiene $\lambda u.y$ che non può essere congruente ad alcun sottotermine nel λ-termine di partenza, rappresentato graficamente da:



mentre la nostra occorrenza è rappresentata da:

Tale occorrenza potrebbe scomparire per effetto di altre riduzioni, ma in tal caso scomparirebbe anche la variabile libera y contrariamente al fatto che nel-

l'albero sia la seconda occorrenza libera di y. In definitiva possiamo concludere che $\Big(\big(y(\lambda x.xzy)\big)_H\Big)_\lambda$ non è debolmente uguale a $y(\lambda x.xzy)$

Teorema. 6.2 *(di equivalenza)*

- *Se X e Y sono termini combinatori, allora:*

$$X = Y \text{ se e soltanto se } X_\lambda = Y_\lambda.$$

- *Se X e Y sono λ-termini, allora:*

$$X = Y \text{ se e soltanto se } X_H = Y_H.$$

- *Se X è termine combinatorio, allora:*

$$(X_\lambda)_H \equiv X.$$

- *Se X è λ-termine, allora:*

$$(X_H)_\lambda \equiv X.$$

Tale teorema "aggiusta" l'esempio precedente.

Dim. 6.2 *No.*

Quindi se consideriamo i termini combinatori come programmi di un linguaggio di programmazione idealizzato con primitive (**I**, **K**, **S**) e i λ-termini come programmi senza primitive (ma con procedure richiamabili con parametri), vediamo che i due linguaggi idealizzati sono equivalenti a patto di considerare tali programmi come non completi dei loro dati (uguaglianza estensionale).

Capitolo 7

Sistemi combinatori alternativi

Abbiamo implicitamente visto che il sistema di termini combinatori basato sulle costanti riducibili **I**, **K**, **S** potrebbe essere sostituito, a meno di uguaglianze estensionali, da quello basato solo da **K** e **S**.

Di sistemi alternativi, in tal senso, ve ne sono parecchi altri. Uno di particolare importanza è il sistema **B**-**C**-**K**-**W**, ove tali costanti riducibili sono definite dai seguenti assiomi:

(B) $\mathbf{B}XYZ \rhd X(YZ)$;

(C) $\mathbf{C}XYZ \rhd XZY$;

(K) $\mathbf{K}XY \rhd X$;

(W) $\mathbf{W}XY \rhd XYY$.

Per vedere l'equivalenza di tale sistema con i precedenti, si potrebbe per esempio usare l'equivalenza con il λ-calcolo, ma basta vedere con i seguenti esempi che si può definire sempre l'astrazione di un termine in modo che valga il teorema di riduzione.

Esempio. 24 *Cerchiamo* $[x, y, z]xz(yz)$ *che, se varrà il teorema di riduzione, ci darà* **S**.

Oss. 7.1 *Se manteniamo nel nuovo sistema la terza clausola dell'astrazione:*

$$[x]Mx \equiv M,$$

con $x \notin M$*, ci basterà costruire un termine* S' *tale che* $S'xyz = xz(yz)$ *e tale che* $x, y, z \notin S'$*. Infatti per la terza clausola varrà che* $S' = [x, y, z]xz(yz)$ *e varrà il teorema di riduzione.*

Gli assiomi precedenti se letti all'incontrario ci permettono tale costruzione. Infatti (B) *consente di eliminare le parentesi sintattiche,* (C) *di scambiare fra loro dei sottotermini,* (K) *di "inventare" termini che non c'erano e* (W) *di eliminare doppie occorrenze (adiacenti nell'applicazione)di termini.*

Vediamo come ottenere S'.

$$\begin{aligned} xz(yz) &= \mathbf{C}x(yz)z \\ &= \mathbf{B}(\mathbf{C}x)yzz \\ &= \mathbf{W}\big(\mathbf{B}(\mathbf{C}x)y\big)z \\ &= \mathbf{W}(\mathbf{BBC}xy)z \\ &= \mathbf{BW}(\mathbf{BBC}x)yz \\ &= \mathbf{B}(\mathbf{BW})(\mathbf{BBC})xyz. \end{aligned}$$

Quindi otteniamo $S' \equiv \mathbf{B}(\mathbf{BW})(\mathbf{BBC})$ e tale S' avrà la proprietà (S).

7.1 Uso del sistema B-C-K-W

L'esprimere un termine combinatorio nel sistema **B-C-K-W** (con la procedura accennata) ha dei vantaggi quando si voglia usare il calcolo combinatorio in Matematica (in particolare in Algebra).

La Matematica solita è (almeno idealmente) fondata sulla Teoria degli Insiemi. I primi sviluppi a partire da tale fondazione sono di tipo estensionale: di un oggetto (di solito funzione) ci si cura di più delle proprietà (che lo collocano in un estensione di oggetti simili) che non di come sia costruito.

Il come è costruito può essere trovato insiemisticamente (con parecchia fortuna) oppure tramite il calcolo dei combinatori "intensionalmente". Nel secondo modo, che comunque può servire ad ispirare il primo, si hanno a disposizione metodi standard (teorema di riduzione).

Per quanto il sistema **I**, **K**, **S**, appaia più semplice da usare rispetto al **B-C-K-W**, esso è meno adatto per ottenere i risultati di cui sopra. Ciò dipende dal fatto che il parallelizzatore **S** ha meno proprietà "categoriche" dei **B**, **C**, **W**. Accenneremo in seguito a tali proprietà. Queste proprietà consentono di trovare proprietà algebriche degli oggetti costruiti.

7.2 Possibile uso di sistemi combinatori alternativi

Se in ogni termine combinatorio del sistema **K**, **S** sostituissimo **S** con il termine che lo rappresenta nel sistema **B-C-K-W**, allora otterremmo un sistema equivalente al secondo. In sostanza avremmo fatto una "sostituzione multipla":

$$[\mathbf{K}/\mathbf{K},\ \mathbf{B}(\mathbf{BW})(\mathbf{BBC})/\mathbf{S}]$$

sulle costanti riducibili anzichè sulle variabili. In pratica tali "sostituzioni" servono a tradurre un programma idealizzato nel primo linguaggio di programmazione (sistema **I**, **K**, **S**) in uno relativo al secondo (sistema **B-C-K-W**).

Pertanto abbiamo un esempio di traduttore di linguaggi di programmazione idealizzati. L'uso di tali "traduttori ideali" può servire a studiare proprietà dei traduttori reali. Può anche servire a definire, in ambito puramente linguaggistico, nozioni equivalenti a quelle note dalla Teoria di Turing (per esempio l'universalità) che sono espresse tramite agenti di calcolo.

7.3 Sistemi di riscrittura

Quale che sia il sistema combinatorio di riferimento, abbiamo una nozione di riduzione espressa, oltre che da regole di deduzione, da assiomi del tipo:

$$\Gamma X' X'' \cdots X^{(n)} \rhd T,$$

ove T è un termine formato dagli $X^{(i)}$.

In realtà per definire una nozione di calcolo "$\rhd$", non è necessario che gli assiomi siano fatti in tal modo[1]; infatti nei linguaggi di programmazione effettivamente usati tale restrizione non esiste. D'altra parte il tipo di risultati che si ottengono con il calcolo combinatorio, per dei linguaggi idealizzati, è molto interessante anche per quelli effettivamente utilizzati.

Per rispondere a tale esigenza sono stati definiti i "sistemi di riscrittura" (rewriting systems) in cui gli assiomi possono essere scritti più liberamente. In tali sistemi si richiede che valga comunque il teorema di Church-Rosser, in modo che la relazione "$\rhd$" abbia quelle carateristiche di "quasi funzionalità" che l'idea di calcolo richiede.

Nota: i sistemi di riscrittura *non* sono imparentati con i "sistemi di riscrittura parallela" che riguardano si l'informatica teorica, ma solo quella algebrica.

Esercizio. 4 *Trovare nel sistema* **B-C-K-W** *un combinatore equivalente al "*Φ*", visto in 2.5.3, tale che:*

$$\Phi XYZU = X(YU)(ZU).$$

[1] Per esempio che la costante riducibile Γ stia in testa o che si abbia un'applicazione binaria

$$
\begin{aligned}
x(yu)(zu) &\overset{(B)}{\equiv} \mathbf{B}xyu(zu)\\
&\overset{(C)}{\equiv} \mathbf{C}(\mathbf{B}xy)(zu)u\\
&\overset{(B)}{\equiv} \mathbf{B}\big(\mathbf{C}(\mathbf{B}xy)\big)zuu\\
&\overset{(W)}{\equiv} \mathbf{W}\Big(\mathbf{B}\big(\mathbf{C}(\mathbf{B}xy)\big)\Big)zu\\
&\overset{(B)}{\equiv} \mathbf{W}\Big(\mathbf{B}\big(\mathbf{BC}(\mathbf{B}x)y\big)\Big)zu\\
&\overset{(B)}{\equiv} \mathbf{BWB}\big(\mathbf{BC}(\mathbf{B}x)y\big)zu\\
&\overset{(B)}{\equiv} \mathbf{BWB}\big(\mathbf{B}(\mathbf{BC})\mathbf{B}xy\big)zu\\
&\overset{(B)}{\equiv} \mathbf{BWB}(\mathbf{BBBCB}xy)zu\\
&\overset{(B)}{\equiv} \mathbf{B}(\mathbf{BWB})\mathbf{BBBCB}xyzu\\
&\overset{(B)}{\equiv} \mathbf{BBBWBBBBCB}xyzu.
\end{aligned}
$$

Togliendo le variabili xyzu si ha un tale Φ'.

Capitolo 8

Tipi e termini con tipo

In Teoria degli Insiemi si considera la nozione di appartenenza, tramite tale nozione "$x \in X$" se ne ottengono altre: $f\colon A \to B$ o $f\colon A^X \to B^X$ etc. Nella logica combinatoria tale appartenenza non c'è. Si ha invece la nozione (operativa) di tipo.

Si assume l'esistenza di una sequenza (non vuota) di *tipi* detti *fondamentali*: α, β, γ, ... Un tipo fondamentale puó essere correttamente inteso come l'indicazione dove trovare qualcosa, per esempio l'etichetta di un supporto dati, il nome di una periferica, etc. Non correttamente (ma utilmente) lo si può interpretare come un insieme di base (con cui poi costruire altri insiemi mediante esponenziazione).

Def. 8.1 *I tipi sono defini dalle seguenti clausole.*

- *Ogni tipo fondamentale è un tipo;*
- *Se α e β sono tipi, allora $F\alpha\beta$ è un tipo.*

Ivi $F\alpha\beta$ indica un'applicazione diversa da quella per i termini combinatori, ma definita allo stesso modo (cioè si scrive F come funzione solo per agevolare la scrittura).

Per attribuire tipi a dei termini ci sono due modi parziali e uno completo[1].

Consideriamo ora il primo modo; consiste nel ridefinire i termini combinatori (si può fare anche per i λ-termini ma non lo vedremo) in modo che vengano generati con un tipo fisso.

Si assume per ogni tipo $\sqcup$ una successione infinita di variabili e per indicare che una variabile x è di tipo $\sqcup$ scriveremo $x^{\sqcup}$. Similmente per ogni costante non riducibile a avremo $a^{\sqcup}$. Per ogni tipo α avremo una costante (riducibile) $\mathbf{I}_\alpha$ alla quale attribuiremo il tipo $F\alpha\alpha$. Per ogni coppia di tipi α, β avremo una costante $\mathbf{K}_{\alpha,\beta}$ a cui attribuiremo tipo $F\alpha(F\beta\alpha)$. Infine per ogni terna di tipo α, β, γ avremo una costante $\mathbf{S}_{\alpha,\beta,\gamma}$ cui attribuiamo tipo $F(F\alpha(F\beta\gamma))(F(F\alpha\beta)(F\alpha\gamma))$.

Si suppone che tutti questi oggetti, che chiamiamo *atomi* siano distinti.

[1] Che non faremo, ma si trova sul testo "non consigliato" e successivo di Hindley e altri.

Def. 8.2 *I termini con tipo sono definiti dalle seguenti clausole.*

- *Ogni atomo è un termine con tipo;*
- *Se U è un termine con tipo $F\alpha\beta$ e V è un termine con tipo α, allora UV è un termine di tipo β.*

Verranno considerati solo termini costruiti come sopra (nel primo modo parziale). Vale la solita definizione di applicazione con le clausole di disgiunzione.

Nota: spesso si hanno tipi complicati come $F\alpha_1(F\alpha_2(\ldots(F\alpha_n\beta)\ldots))$, tale scrittura si semplifica nel seguente modo: $F_n\alpha_1\alpha_2\cdots\alpha_n\beta$. Per esempio possiamo scrivere $F_2\alpha\beta\alpha$ come tipo per $\mathbf{K}_{\alpha,\beta}$ e $F_3(F_2\alpha\beta\gamma)(F\alpha\beta)\alpha\gamma$ come tipo di $\mathbf{S}_{\alpha,\beta,\gamma}$.

Lemma. 8.1 *Per ogni tipo α, $\alpha \not\equiv F\alpha\beta$, $F\beta\alpha$.*

Dim. 8.1 *Nel caso α sia tipo fondamentale ció deriva dall'ipotesi di disgiunzione implicita nella definizione: $F\alpha\beta$ è un'applicazione. Altrimenti si procede per induzione binaria secondo la definizione di tipo (in sostanza si ritorna al caso fondamentale).*

Def. 8.3 *La sostituzione $[Y^\alpha/x^\alpha]X$ è definita come per il λ-termini, ma con la restrizione che la variabile e il termine sostituito abbiano lo stesso tipo.*

Def. 8.4 *Diremo che un termine combinatorio X si riduce (debolmente) a un termine combinatorio Y e scriveremo $X \rhd Y$ quando ciò può essere dedotto dagli assiomi:*

(I_α) $\mathbf{I}_\alpha X^\alpha \rhd X^\alpha$.

($K_{\alpha,\beta}$) $\mathbf{K}_{\alpha,\beta}X^\alpha Y^\beta \rhd X^\alpha$.

($S_{\alpha,\beta,\gamma}$) $\mathbf{S}_{\alpha,\beta,\gamma}X^{F\alpha(F\beta\gamma)}Y^{F(F\alpha\beta)(F\alpha\gamma)}Z^\alpha \rhd XZ(YZ)$ *che tipo avrà?*

Mediante le regole di deduzione:

- *Assioma di riflessività.*
- $X^{F\alpha\beta} \rhd Y^{F\alpha\beta}$ *implica* $X^{F\alpha\beta}Z^\alpha \rhd Y^{F\alpha\beta}Z^\alpha$.
- $X^\alpha \rhd Y^\alpha$ *implica* $Z^{F\alpha\beta}X \rhd Z^{F\alpha\beta}Y^\alpha$.
- *Transitività.*

Sostituendo "=" al simbolo "$\rhd$" e aggiungendo la regola di simmetria si definisce l'uguaglianza debole (=).

Teorema. 8.1 *Se $X = Y$, allora X e Y hanno lo stesso tipo.*

Dim. 8.2 *Applichiamo le definizioni di uguaglianza (=) considerando dapprima gli assiomi.*

(I_α) Immediata dalla clausola di composizione.

($K_{\alpha,\beta}$) Quasi immediata. $\mathbf{K}_{\alpha,\beta}$ ha tipo $F\alpha(F\beta\alpha)$ e X ha tipo α per cui $\mathbf{K}_{\alpha,\beta}X^\alpha$ ha tipo $F\beta\alpha$ e $\mathbf{K}_{\alpha,\beta}X^\alpha Y^\beta$ avrà tipo α come il secondo membro dell'applicazione (X^α).

($S_{\alpha,\beta,\gamma}$) Consideriamo dapprima il secondo membro $XZ(YZ)$. Per essere un termine combinatorio con tipo (che indicheremo con γ) il primo componente XZ dovrà avere tipo $F\beta\gamma$ per qualche β. Affinchè ciò sia possibilie X dovrà avere tipo $F\alpha(F\beta\gamma)$ per qualche α. Similmente affinchè YZ abbia tipo β, Y dovrà avere tipo $F\delta\beta$ ove δ sia il tipo di Z, ma per la costruzione di XZ tale δ è α, cioè Y ha tipo $F\alpha\beta$. In definitiva:

$$\begin{array}{rl} Z & \text{ha tipo } \alpha. \\ Y & \text{ha tipo } F\alpha\beta. \\ X & \text{ha tipo } F\alpha(F\beta\gamma). \\ XZ(YZ) & \text{ha tipo } \gamma. \end{array}$$

Passiamo ora al primo membro dell'uguaglianza, che dovrà risultare di tipo γ. Avendo assegnato $F(F\alpha F\beta\gamma)(F(F\alpha\beta)(F\alpha\gamma))$ a $\mathbf{S}_{\alpha,\beta,\gamma}$ per la clausola di composizione (e per il tipo di X) si ha che $\mathbf{S}_{\alpha,\beta,\gamma}X$ è termine combinatorio con tipo che risulta $F(F\alpha\beta)(F\alpha\gamma)$. Da qui troviamo il tipo di $\mathbf{S}_{\alpha,\beta,\gamma}XY$ allo stesso modo, cioè $F\alpha\gamma$. Come prima si trova il tipo desiderato di $\mathbf{S}_{\alpha,\beta,\gamma}XYZ$, ovvero γ.

Infine è immediato controllare che tale conservazione del tipo viene rispettata dalle regole di deduzione.

Def. 8.5 *Data una variabile x^α e un termine X^β definiamo l'astrazione $[x]X$ con le segenti clausole:*

- $[x]x \equiv \mathbf{I}_\alpha$
- $[x]M \equiv \mathbf{K}_{\beta,\alpha}M$ *ove* $x \notin M$ *e* β *sia il tipo di* M
- $[x]Ux \equiv U$ *ove* $x \notin U$
- $[x]UV \equiv \mathbf{S}_{\alpha,\delta,\beta}([x]U)([x]V)$ *ove* δ *sia il tipo di* V *e* $F\delta\beta$ *il tipo di* U.

Lemma. 8.2 *$[x^\alpha]X^\beta$ ha tipo $F\alpha\beta$.*

Dim. 8.3 *Per induzione combinatoria su X (seguendo il raggruppamento della definizione).*

a) $[x^\alpha]X^\beta \equiv \mathbf{I}_\alpha$ che ha tipo $F\alpha\alpha$, ma $X^\beta \equiv x^\alpha$, cioè $\beta \equiv \alpha$.

b) $[x^\alpha]X^\beta \equiv \mathbf{K}_{\beta,\alpha}X^\beta$ che, per clausole combinatorie[2] ha tipo $F\alpha\beta$.

[2] $\mathbf{K}_{\beta,\alpha}$ ha tipo $F\beta(F\alpha\beta)$.

c) $[x^\alpha]X^\beta \equiv U$ *poiché* $X^\beta \equiv Ux$, *avendo* x *tipo* α, *per le solite clausole* U *deve avere tipo* $F\beta\alpha$.

d) $[x^\alpha]X^\beta \equiv \mathbf{S}_{\alpha,\delta,\beta}([x]U)([x]V)$.

In questo caso servono le premesse di induzione $([x]U)^{F\alpha(F\delta\beta)}$ *e* $([x]V)^{F\alpha\delta}$. *Da tali premesse, poichè* $\mathbf{S}_{\alpha,\delta,\beta}$ *ha tipo* $F(F\alpha(F\delta\beta))(F(F\alpha\delta)(F\alpha\beta))$ *si ha dapprima che* $\mathbf{S}_{\alpha,\delta,\beta}([x]U)$ *è termine combinatorio con tipo che risulta* $F(F\alpha\delta)(F\alpha\beta)$ *e poi che* $\mathbf{S}_{\alpha,\delta,\beta}([x]U)([x]V)$ *è termine combinatorio di tipo* $F\alpha\beta$.

Oss. 8.1 *Si noti che da tale lemma si ha che per un'astrazione multipla (iterata come al solito) il tipo di* $[x^{\alpha_1}, x^{\alpha_2}, \ldots, x^{\alpha_n}]X^\beta$ *è* $F_n\alpha_1\alpha_2\cdots\alpha_n\beta$ *(giustificazione della semplificazione* F_n*).*

8.1 Risultati

Il sistema dei termini con tipo può essere considerato come una restrizione di un sistema **I**, **K**, **S** (espanso agli $\mathbf{I}_\alpha$, $\mathbf{K}_{\alpha,\beta}$, $\mathbf{S}_{\alpha,\beta,\gamma}$) poiché certe applicazioni non sono ammesse. Pertanto è abbastanza ovvio che tutti i risultati per **I**, **K**, **S** valgano ancora (con dimostrazioni leggermente variate). Non vale invece il teorema (insiemistico) di inseparabilità ricorsiva. Infatti oltre ai predetti risultati si ha il seguente teorema.

Teorema. 8.2 *Ogni termine con tipo ha forma normale.*

Dim. 8.4 *No (idea: mancanza di universalitá).*

Corollario 8.1 *L'uguaglianza fra termini con tipo è decidibile.*

Dim. 8.5 *Segue dal terzo corollario di Church-Rosser II.*

Nota: Si può pensare di fare in "parallelo" tutte le riduzioni possibili e di controllare ogni volta se abbia una forma normale che sappiamo per il teorema esistere.

Ciò permette di verificare l'uguaglianza per programmi idealizzati. Risultati di questo tipo sono stati ottenuti anche per linguaggi di programmazione non idealizzati (reali). Ciò ha dato luogo allo sviluppo della programmazione strutturale o modulare.

Con opportuni linguaggi di programmazione si costringe il programmatore a strutturare il programma in moduli che abbiano delle configurazioni d'entrata ed uscita ben precisate e compatibili. Non solo tali programmi sono a "terminazione garantita" ma in fase di compilazione il compilatore si accorge di errori in un'ampia classe di errori, rilevabili così *automaticamente* prima dell'esecuzione (ovviamente si perde l'universalità).

8.2 Assegnazione di tipi a termini senza tipo

Considerando che i precedenti termini con tipo non sono tutti quelli senza tipo (estesi), ci si può chiedere quando un termine senza tipo corrisponda ad uno con tipo. A tal fine (re)introduciamo i termini senza tipo ma con costanti che comprendano i tipi fondamentali ed F (oltre ad $\mathbf{I}$, $\mathbf{K}$, $\mathbf{S}$). Pertanto avremo come termini anche $F\alpha\beta$ (inteso come al solito $(F\alpha)\beta$), $F\alpha$, F, etc[3]. Nella definizione di assegnazione introdurremo la notazione:

(1) $\quad \vdash \alpha X$

per indicare che un tipo α può essere assegnato ad un termine X (infatti nelle applicazioni logiche, che non faremo, ció si leggerà "è dimostrato che X ha la proprietà α). Si noti che, in accordo con l'interpretazione (scorretta ma utile) insiemistica dei tipi, ció può essere letto come X appartiene ad α.

Esempio. 25 $\vdash F\alpha\alpha\mathbf{I}$ *corrisponde insiemisticamente a* $\mathbf{I} \in \alpha^\alpha$ *cioè* $\mathbf{I}\colon \alpha \to \alpha$.

L'appartenenza ad un insieme è (classicamente) la soddisfazione di una proprietà (insieme $\leftrightarrow$ proprietà). Una scrittura come la (1) sarà detta *enunciato*, mentre X sarà detto il *soggetto* ed α il *predicato*.

Def. 8.6 *Diremo* base *una sequenza di enunciati come la* (1).

Def. 8.7 *Diremo che un tipo* α *è* assegnato (debolmente) *a un termine combinatorio* X *e scriviamo* $\vdash \alpha X$ *quando ciò può essere dedotto dai seguenti assiomi:*

(FI) $\quad \vdash F\alpha\alpha\mathbf{I}$ *per ogni tipi* α

(FK) $\quad \vdash F\alpha(F\beta\alpha)\mathbf{K}$ *per ogni tipo* α, β

(FS) $\quad \vdash F\big(F\alpha(F\beta\gamma)\big)\big(F(F\alpha\beta)(F\alpha\gamma)\big)$ *per ogni tipo* α, β, γ

mediante la regola:

(F) $\quad \vdash F\alpha\beta U$ *e* $\vdash \alpha V$ *implica* $\vdash \beta(UV)$

Nota: tale scrittura viene di solito abbreviata in:

$F\alpha\beta U,\ \alpha V\ \vdash \beta(UV)$,

e similmente per enunciati di una base.

Con tale definizione possiamo assegnare tipi solo a dei combinatori. Nel caso generale si considera una base B e si definisce l'assegnazione $B \vdash \alpha X$ come nella definizione precedente però aggiungendo B agli assiomi.

Def. 8.8 *Consideriamo una base* B *in cui non occorrano variabili, diremo che un termine* X *è* stratificato relativamente *a* B *se esiste un tipo* α *tale che:*

$$B \vdash \alpha X.$$

Se B *è vuota diremo che* X *è* stratificato *(non stiamo usando una base).*

[3] Ció serve ad applicazioni relative alla Logica Matematica e alla Teoria delle Strutture di Dati, che non vedremo.

Si può vedere che tale definizione riguarda solo degli X "costanti", cioè privi di variabili.

Def. 8.9 *Nel caso in cui le variabili $x_1, x_2, \ldots, x_n$ occorrano in X diremo che X è stratificabile relativamente a B qualora esistano dei tipi $\xi_1, \xi_2, \ldots, \xi_n$ ed α tali che:*

$$B, \xi_1 x_1, \xi_2 x_2, \ldots, \xi_n x_n \ \vdash \alpha X$$

Se B è vuota X si dice stratificabile.

In pratica determinare se un termine è stratificato (o straficabile) relativamente a B significa cercare di trovarne il tipo e vedere cosa succede. Per esempio consideriamo $X \equiv \mathbf{KS}$.

Da (FS) vediamo che possiamo applicare la regola (F) qualora in (FK) si prenda come α un tipo α' come ivi indicato.

$$\vdash F \underbrace{\Big(F\big(F\alpha(F\beta\gamma)\big)\big(F(F\alpha\beta)(F\alpha\gamma)\big)\Big)}_{\alpha'} \big(F\beta' \underbrace{(\ldots)}_{\alpha'}\big)\mathbf{K} \qquad \vdash \underbrace{(\ldots)}_{\alpha'}\mathbf{S}.$$

Usando la regola (F)

$$\vdash F\beta'\big(\underbrace{(\ldots)}_{\alpha'}\big)(\mathbf{KS}).$$

Quando il termine è ancora un po' più complicato, per esempio per $X \equiv \mathbf{B} \equiv \mathbf{S}(\mathbf{KS})\mathbf{K}$, tali esplosioni dei sottotipi necessari per applicare la (F) diventano proibitive (almeno manualmente). Ciò avviene nonostante che il tipo di $\mathbf{B}$ sia relativamente semplice.

$$\vdash F(F\beta\gamma)\big(F(F\alpha\beta)(F\alpha\gamma)\big)\mathbf{B}$$

D'altra parte a tale tipo di $\mathbf{B}$ si arriva molto più facilmente utilizzando l'interpretazione insiemistica ($\mathbf{B} \leftrightarrow$ composizione invertita) per la quale:

$$\mathbf{B}\colon C^B \to (C^A)^{B^A}.$$

Pertanto ci si può chiedere se in un'assegnazione $\vdash \alpha X$ non si possa utilizzare (questa volta rigorosamente) lo stesso procedimento. Tale procedimento insiemistico era basato sull'uso dell'astrazione (se un argomento di f stà in A e f_a stà in B allora $f\colon A \to B$). Vedremo in seguito che ciò vale.

Esempio. 26 *Consideriamo* **SII**. *Esso è "cattivo" nel senso che per ogni X $\mathbf{SII}X = XX$, che insiemisticamente non ha un'interpretazione pulita. Verifichiamo se è stratificato. Se lo fosse ci sarebbe α tale che:*

(2) $\quad \vdash \alpha(\mathbf{SII})$*;*

ma la (2) *può provenire solo dalla (F) poiché il termine è composto. Allora dovrebbe esistere un β tale che:*

$$\vdash F\beta\alpha(\mathbf{SI}),\ \vdash \beta\mathbf{I}.$$

analogamente dovrebbe esserci δ tale che:

(3) $\vdash F\delta(F\beta\alpha)\mathbf{S}$;

$\vdash \delta\mathbf{I}$.

Ma la (3) *deve essere un caso di* (FS) *perciò* $\delta \equiv F\alpha'(F\beta'\gamma')$, $\beta \equiv F\alpha'\beta'$ *e* $\alpha \equiv F\alpha'\gamma'$ *per qualche* α', β', γ'. *Tuttavia dell'assegnazione di* β *ad* $\mathbf{I}$ $(\vdash \beta\mathbf{I})$ *sappiamo che* $\alpha' \equiv \beta'$. *Da ció e da* $\vdash \delta\mathbf{I}$ *si ha che (essendo* $\alpha' \equiv F\beta'\gamma'$*)* $\beta' \equiv F\beta'\gamma'$. *Quindi se vi fosse* α *tale che* $\vdash$(**SII**) *avremmo un* β' *che coincide con un suo sottotipo. In definitiva* **SII** *non è stratificato.*

Ció può far sembrare che sia collegato con la proprietà precedente (**SII** era combinatore di autoapplicazione). I due esempi successivi chiariscono il problema.

Esempio. 27 KK *è stratificato. Infatti per la* (FK) *abbiamo le due premesse della deduzione:*

$$\vdash F\alpha(F\beta\alpha)\mathbf{K},$$

$$\vdash F\big(F\alpha(F\beta\alpha)\big)\Big(F\beta'\big(F\alpha(F\beta\alpha)\big)\Big)\mathbf{K}.$$

Tramite la regola (F) *otteniamo:*

$$\vdash F\beta'\big(F\alpha(F\beta\alpha)\big)(\mathbf{KK}).$$

Esempio. 28 xx *per ogni variabile* x *non è stratificabile. Infatti se lo fosse avremmo* ξ *ed* α *tali che* $\xi x \vdash \alpha(xx)$. *Essendo* xx *composto ció dovrebbe provenire dalla* (F) *per cui*

$$\xi x \vdash F\xi\alpha x,$$

$$\xi x \vdash \xi x.$$

Tramite la regola (F) *otteniamo:*

$$\xi x \vdash \alpha(xx).$$

Mentre la seconda premessa è valida, la prima ci dice che $\xi \equiv F\xi\alpha$ *che è impossibile.*

Nota: il procedimento visto negli esempi può essere generalizzato in un algoritmo che consente di terminare con esito positivo o negativo (algoritmo ricorsivo) se un termine è stratificato (o stratificabile) relativamente a delle basi (sufficientemente generali).

Pertanto nell'insieme $\mathcal{T}$ dei termini combinatori, il sottoinsieme $\mathcal{T}'$ dei termini stratificati (o stratificabili)[4] è ricorsivo. Sostanzialmente questa è la ragione per cui nella programmazione strutturata la compilazione è in grado di trovare i "bachi" che con altri linguaggi di programmazione non era possibile trovare.

[4]Nel quale l'uguaglianza debole diventa decidibile.

Teorema. 8.3 *Se B è una base senza variabili e se per ogni variabile x e termine X $B, \alpha x \vdash \beta X$, allora:*

$$B \vdash F\alpha\beta([x]X).$$

Dim. 8.6 *Come per il lemma analogo dei termini con tipo.*

Possiamo quindi utilizzare l'astrazione per calcolare i tipi (come si faceva nell'interpretazione insiemistica).

Teorema. 8.4 (riduzione del soggetto) *Sia B una base in cui ogni soggetto sia irriducibile rispetto alla "$\triangleright$" e in cui esso non cominci con un combinatore. Se $B \vdash \alpha X$ e $X \triangleright Y$ allora:*

$$B \vdash \alpha Y.$$

Dim. 8.7 *No.*

Si noti che nella terminologia logica introdotta ció significa che una proprietà espressa dal predicato α a proposito del soggetto X viene sempre ereditata da un soggetto Y cui si pervenga tramite il nostro processo di calcolo "$\triangleright$" partendo da X. Di tale teorema esiste anche l'estensione relativa al processo di calcolo "$\succ$" (riduzione forte che definisce l'uguaglianza estensionale). Non vale tuttavia tale conservazione per "=" (sia debole che forte) poiché nella premessa del teorema non si può sostituire la clausola $X \triangleright Y$ con $Y \triangleright X$.

Esempio. 29 *Sia $Y \equiv$* **SKSI**.

Allora si può vedere che (dalla base vuota) $\vdash F\delta\delta Y$ vale solo per dei δ composti. Invece considerando che:

$$Y \triangleright \mathbf{KI}(\mathbf{SI}) \triangleright \mathbf{I} \equiv X.$$

abbiamo che $\vdash F\delta\delta X$ per ogni δ.

In modo analogo si può vedere che per $Y \equiv \mathbf{SIII} \triangleright \mathbf{II}(\mathbf{II}) \triangleright \mathbf{I} \equiv X$, mentre X è ovviamente stratificata, Y non può esserlo — poiché se lo fosse lo sarebbe **SII** che abbiamo visto non esserlo (problema espansione del soggetto).

Consideriamo l'enunciato del teorema d'astrazione nel caso semplificato di base vuota (che ovviamente soddisfa le condizioni sulle variabili).

$$\vdash \alpha x,\ \vdash \beta X \text{ implica } \vdash F\alpha\beta([x]X).$$

(non c'e' $\vdash \alpha x$ nella conclusione)

In sostanza abbiamo dedotto $F\alpha\beta$ per $[x]X$ senza una base, ma usando una premessa $\vdash \alpha x$ della deduzione che è "scomparsa". Quindi per usare tali deduzioni si devono "cancellare" tali premesse.

Da tale asserto nasce la seguente definizione.

Def. 8.10 *Diremo che, assegnata una base B in cui non occorrano variabili, possiamo assegnare* naturalmente *il tipo α ad un termine combinatorio X e scriveremo $B \vdash^T \alpha X$ quando ció può essere dedotto dagli enunciati in B tramite le regole:*

(F) $B \vdash^T F\alpha\beta X$ *e* $B \vdash^T \alpha Y$ *implica* $B \vdash^T \beta(XY)$.

(F_i) $B, \alpha x \vdash^T \beta X$ *implica* $B \vdash^T F\alpha\beta([x]X)$ *ove in tutte le premesse dei passi induttivi precedenti si cancellano le assegnazioni provvisorie.*

Si noti che non vi sono assegnazioni per le costanti riducibili. Nel caso di base vuota "si parte da niente", nel senso che le assegnazioni da cui si parte devono essere cancellate. Si noti, inoltre, che la clausola di cancellazione è automaticamente soddisfatta nel caso in cui l'assegnazione provvisoria da cancellare non esista.

Nel caso della vecchia assegnazione "$\vdash$" la deduzione di un tipo α per X consisteva in un albero binario (isomorfo a quello di X) le cui foglie corrispondono alle assegnazioni per le costanti riducibili e agli enunciati in B, mentre gli altri nodi corrispondevano ad impieghi della regola (F).

Ora tale albero per una deduzione "$\vdash^T$", oltre a non avere le foglie per le costanti riducibili, ha dei nodi corrispondenti agli impieghi della (F_i). Tali nodi sono unari perchè dalla singola assegnazione (genitore) ad X si passa all'assegnazione per $[x]X$. Quindi l'albero ora è *al più* binario. Inoltre occorre ad ogni uso della (F_i) fare le cancellazioni. Per fare ció si marca ogni impiego delle (F_i) con un'etichetta da ripetere sull'eventuale assegnazione da cancellare.

Teorema. 8.5 *Per ogni base B in cui non occorrono variabili si ha che:*

$$B \vdash \alpha X \text{ se e solo se } B \vdash^T \alpha X \text{per ogni } \alpha \text{ ed } X.$$

Dim. 8.8 *La parte "se" deriva dal teorema d'astrazione. Per la parte "solo se" basta mostrare che gli assiomi (FI), (FK) ed (FS) possono essere dedotti col nuovo sistema. Saltiamo i primi due, che sono abbastanza banali (per (FK) si cancella una premessa che non esiste), consideriamo (FS). Notiamo che $[z]xz(yz) \equiv \mathbf{S}xy$ (dall'ultima clausola per l'astrazione). Perció useremo la (F) con assegnazione provvisoria fino a raggiungere $xz(yz)$ e poi useremo (F_i).*

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash F\alpha(F\beta\gamma)x \;\;[3] \qquad \vdash \alpha z \;\;[1]}{F\beta\gamma(xz)}(F)}{\vdash F\alpha\beta y \;\;[2] \qquad \vdash \alpha z \;\;[1]}(F)}{\vdash \beta(yz)}(F)}{\vdash F\alpha\beta y \qquad \vdash \beta(yz)}(F)}{\vdash \gamma\big(xz(yz)\big)}(Fi)\quad[1]}{\vdash F\alpha\gamma \qquad [z]xz(yz)}(Fi)\quad[2]$$

$$\dfrac{\vdash F(F\alpha\beta)(F\alpha\gamma) \qquad \mathbf{S}x \equiv [y]\mathbf{S}xy}{\vdash F\big(F\alpha(F\beta\gamma)\big)\big(F(F\alpha\beta)(F\alpha\gamma)\big) \qquad \mathbf{S}}(Fi)\quad[3]$$

Capitolo 9

Applicazioni matematiche

9.1 Logica Matematica

Il calcolo dei combinatori serve allo studio di sistemi di Logica Formale. Si introducono nuove costanti per esprimere ad esempio i quantificatori, l'implicazione, etc. Le proprietà combinatorie ci danno proprietà di tali sistemi (coerenza, completezza, etc.). La deduzione precedente ci fornisce il sistema di "deduzione naturale di Guntzen".

9.2 Algebra (e informatica algebrica)

Premessa. Molte proprietà algebriche discendono da proprietà insiemistiche delle funzioni. In particolare da quelle delle classi di funzioni iniettive ($\rightarrowtail$), suriettive ($\twoheadrightarrow$), e biettive ($\rightarrowtail\!\!\!\!\twoheadrightarrow$) su qualche insieme.

Tali classi sono ciascuna chiusa rispetto alla composizione funzionale ed hanno elementi neutri a destra e a sinistra (tramite opportune restrizioni ciò dà luogo ad importanti *categorie*).

Per tali applicazioni il sistema **B-C-K-W** è particolarmente utile. Analizziamo le nostre costanti riducibili.

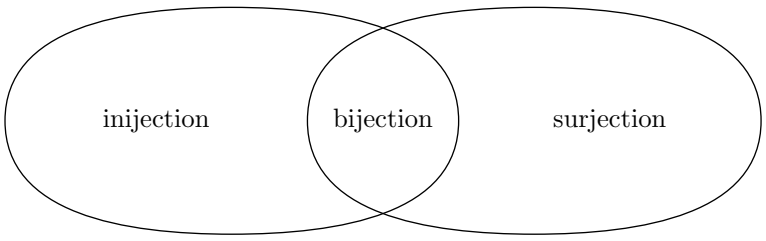


Figura 9.1: Classi delle funzioni iniettive, biiettive e suriettive.

9.2.1 Combinatore B

Dopo applicazione a sinistra "conserva" tali classi di funzioni qualora lo si intenda nell'interpretazione insiemistica. Ovvero:

$$f\colon A \mapsto\!\!\!\!\rightarrow B \text{ implica } \mathbf{B}_X f\colon A^X \mapsto\!\!\!\!\rightarrow B^X;$$

ove $\mathbf{B}_X$ indica l'operatore funzionale insiemistico di "tipo":

$$\mathbf{B}_X\colon B^A \to (B^X)^{A^X}.$$

Si può dimostrare facilmente anche che:

$$f\colon A \twoheadrightarrow B \text{ implica } \mathbf{B}_X f\colon A^X \twoheadrightarrow B^X.$$

Tale $\mathbf{B}$ è di composizione invertita. Vedremo poi quello di composizione diretta $\mathbf{CB}$.

9.2.2 Combinatore C

Insiemisticamente $\mathbf{C}\colon (G^B)^A \to (G^A)^B$ e si vede facilmente che:

$$\mathbf{C}\colon (G^B)^A \mapsto\!\!\!\!\twoheadrightarrow (G^A)^B.$$

Inoltre con la solita notazione per iterazione $\mathbf{C}^2 = \mathbf{I}$. Tale $\mathbf{C}$ compare in proprietà algebriche dette universalità dei prodotti diretti (che oltre all'esponenziazione insiemistica coinvolgono anche il prodotto insiemistico). Componendo con $\mathbf{B}$ si ha l'operatore di composizione diretta.

$$\mathbf{CB}_X\colon A^X \to (B^X)^{B^A}.$$

Tale operatore è ben noto in algebra (monoidi, sottomonoidi, categorie e composizione funzionale) ma è importante anche al di fuori di tale interpretazione.

Esempio. 30 *Rappresentazione matriciale.*

Sia $E \subseteq A^A$ l'insieme degli indomorfismi di uno spazio vettoriale su A. Una funzione $b\colon X \to A$ è una base dello spazio se e soltanto se

$$\mathbf{CB}_X b\colon E \mapsto\!\!\!\!\twoheadrightarrow A^X$$

cioè se e soltanto se per ogni $h \in E$, $\mathbf{CB}_X bh = \mathbf{B}_X hb = h \circ b\colon X \to A$ rappresenta biunivocamente h.

In altre parole $\mathbf{CB}$ ci permette di rappresentare endomorfismi tramite una *base* b (sistema di riferimento) con delle *matrici* in A^X ove per ogni m tale che:

$$m\colon X \to A$$

($m = h \circ b$) m_x è la colonna x-esima della matrice. La differenza con la presentazione geometrica di tali nozioni è che abbiamo usato dei combinatori. Siamo

andati a vedere dal punto di vista operativo, o "intensionale", come vanno le cose. Ciò fa si che tale costruzione funzioni anche al di fuori degli spazi vettoriali. Vi sono infatti algebre che non sono spazi vettoriali (monoidi, gruppi, reticoli, etc.) considerate in Matematica e molte altre considerate in Informatica Algebrica. Ciò fa si che l'implementazione dell'Algebra Lineare (nel senso di Geometria Analitica) possa "transitare" su tutte le altre algebre. Questo realizza la congettura di A. N. Whitehead su di un metodo d'interpretazione unica per tutte le algebre derivate dall'Algebra Lineare. Mediante tale impostazione sono stati riscritti problemi che l'Algebra aveva posto ma non risolto.

9.2.3 Combinatore K

Nell'interpretazione insiemistica è iniettivo:

$$\mathbf{K}\colon A \rightarrowtail A^B;$$

cioè: $\mathbf{K}_a = \mathbf{K}_{a'}$ implica $a = a'$.

9.2.4 Combinatore W

Nell'interpretazione insiemistica è suriettivo:

$$\mathbf{W}\colon (B^A)^A \twoheadrightarrow B^A;$$

cioè per ogni $g\colon A \to B,$ esiste $f\colon A \to B^A$ tale che $g = \mathbf{W}f$.

Capitolo 10

Applicazioni informatiche

In questo capitolo vedremo le varie applicazioni dei termini combinatori in ambito informatico. Salteremo, avendola già considerata nel capitolo dei termini con tipo, la *programmazione strutturata*, mentre vedremo in breve il linguaggio di programmazione LISP.

10.1 Introduzione al LISP

Storia: Nel 1960 McCarty (allievo di Church) propose un linguaggio di programmazione basato sul λ-calcolo. Nel 1962 abbiamo la prima implementazione (funzionante). Fino ad allora la programmazione veniva fatta esclusivamente in Assembler. Nel 1964 furono proposti dapprima l'Algol, poi il Fortran; linguaggi orientati al calcolo numerico. Il Fortran era sostanzialmente solo un Assembler potenziato con un traduttore di formule e divenne operativo nel 1968. L'Algol era più elevato, ma fu "ucciso" commercialmente a causa della "non linearità"[1] del mercato. Tuttavia l'Algol generò altri linguaggi di tipo imperativo.

Il LISP è orientato all'elaborazione simbolica (pur essendo universale), i campi d'applicazione che vedremo danno un'idea di tale tipo di elaborazione; i problemi relativi a questi furono dapprima risolti con il LISP, ma in seguito questo fu sostituito da strumenti specifici nella maggior parte dei casi. Sostanzialmente il LISP è un liguaggio per problemi innovativi.

10.1.1 Campi d'applicazione LISP

- Verifica automatica di dimostrazioni (formalizzate).

[1] Per mercato non lineare si intende un mercato che non segue le regole "canoniche", in cui un prodotto migliore o conveniente ha più successo di uno peggiore o più costoso, ma al contrario segue regole dettate dalla facilità di apprendimento o altri fattori.

- Inferenza induttiva su sequenze. Ad esempio: Qual'è il numero successivo a 5 2 7 9 16 ?

- Calcoli in fisica delle particelle (gruppi).

- Compilatori.

- Compile-compiler. Generano almeno in parte i compilatori partendo da specifiche del linguaggio di programmazione.

- Pattern matching. Trova somiglianze all'interno di due sequenze di simboli.

- Circuiti elettrici.

- Programmazione per calcolo simbolico integro-differenziale. Fino alla fine degli anni '60 si faceva solo calcolo numerico puro, ora esistono pacchetti specifici ad esempio Maple.

- Giochi. Sia deterministici (dama, scacchi, etc) che non (carte).

- Generazione di simulatori.

- Generazione di codici di compressione o di protezione.

- Linguistica.

- Information-retreval, Data-mining.

- Generazione di equazioni diofantine[2] (teorema di superincompletezza di Chaitin).

10.1.2 Idee di base

- λ-termine $\Longleftrightarrow$ S-espressione + λ-astrazione.

- Omogeneità tra dati e procedure. Conseguenze: unicità della struttura di memoria, generabilità di programmi o procedure in esecuzione.

10.1.3 Differenza dal λ-calcolo

- Notazione.

- Aumento delle primitive (utilizzo registri aritmetici, primitive di Ingresso/Uscita, primitive complesse tipo funzioni trigonometriche, etc.).

- Meta-primitive ("define" per richiamo procedure).

- Restrizioni su λ-conversioni.

- Istruzioni per governo sequenzialità di esecuzione.

[2] A variabili intere.

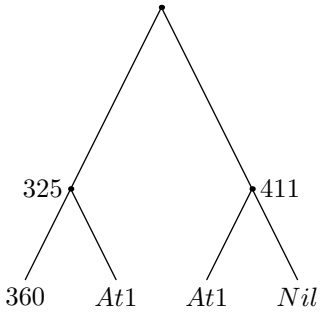


Figura 10.1: Rappresentazione grafica della memoria nell'esempio 31

10.1.4 Organizzazione memoria LISP

Almeno idealmente la memoria LISP (e la relativa notazione) è basata su alberi binari. Essi possono essere implementati mediante *celle* di memoria liberamente sparse nella memoria ad accesso diretto. Ogni cella è costituita almeno idealmente da due campi contenenti di regola indirizzi di altre celle (o di tavole di "atomi").

Esempio. 31 *Indirizi hardware.*

indirizzo	*contenuto*	
1	*xxx*	*xxx*
2	*xxx*	*xxx*
⋮	⋮	
325	*360*	*923*
⋮	⋮	
360	*325*	*411*
⋮	⋮	
411	*923*	*924*
⋮	⋮	
923	*At1*	*Nil*
924	*Nil*	*Nil*

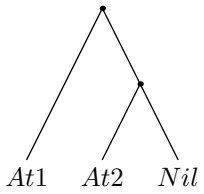
Possiamo pensare agli atomi come ai campi di una tabella che inizi in 900, la foglia 360 invece non è un atomo e ció può servire a (ri)-ciclare sulla cella 360. Per gli atomi il contenuto non è un indirizzo, anzi si può allora semplificare l'interpretazione dicendo che la cella che indirizza un atomo contiene nel campo corrispondente l'atomo stesso. Cioè nell'esempio 31 si deve vedere in 411 la cella

At1	Nil

trattandosi di una struttura a *puntatori* gode dei vantaggi di tali strutture: facile aggiornabilità, facile esecuzione di operazioni strutturali.

Notazione. 2 *Per indicare tali strutture si usano le* S-espressioni. *Si ottengono abbinando col simbolo "." (punto) altre S-espressioni o atomi (così come nella definizione dei termini combinatori ma con il punto che sostituisce l'accostamento).*

Esempio. 32 *Nel caso di:*



avremo $At1.(At2.Nil)$.

Col crescere dell'albero l'S-espressione si complica e aumentano le parentesi, si introduce allora una notazione secondaria: quella di lista. Le liste sono sostanzialmente parole (non alberi), cioè sono generate da una parola vuota (corrispondente all'atomo Nil) e dall'abbinamento di una parola con una lettera dell'alfabeto di atomi diversi da Nil.

Nell'esempio precedente (31) l'alfabeto è fissato per costruire delle parole, tuttavia in generale, si usa come "alfabeto" anche l'insieme delle liste. Per esempio si ha anche la "parola"

$$At1.(\underline{(At2.\underline{(At1.Nil)})}.Nil)$$

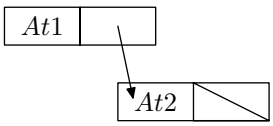
ove le "lettere" sottolineate non sono atomi. Pertanto le liste non sono propriamente parole, ma conviene considerarle tali per ricordarsi che necessitano dell'atomo Nil, che funziona come termine di lista e corrisponde al generatore delle parole.

Per le liste si usa omettere le parentesi associate a destra (non a sinistra come i termini combinatori), rappresentandole come le parole (con l'accostamento sostituito da "$\not{b}$" o da ".") e quindi omettendo Nil. Nel primo esempio scriveremo $At1\ At2$ e nel secondo esempio scriveremo $At1\ (At2\ At1)$. Si noti che questa è una lista diversa da quella indicata con $At1\ At2\ At1$ che corrisponderebbe all'S-espressione $At1.(At2.(At1.Nil))$.

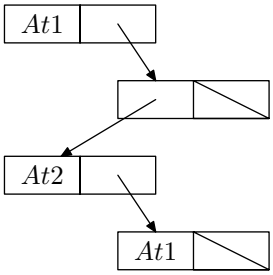
10.2 Rappresentazione grafica di S-termini

[Rap. grafica di S-term.] Per rappresentare graficamente gli S-termini si usano alberi binari con nodi costituiti da una doppia cella: | ind-1 | ind-2 |. Ove però $ind-1$ e $ind-2$ sono graficamente sostituiti da un nome di atomo nel caso che siano indizizzi di un atomo diverso da Nil, da una barra nel caso dell'atomo Nil o da una freccia che punta ad un'altro nodo altrimenti.

Esempio. 33 *Ecco la rappresentazione grafica dell'esempio 32:*



Esempio. 34 *Di seguito vediamo la reppresentazione grafica della parola vista sopra:* $At1.((At2.(At1.Nil)).Nil)$



Tale ultima rappresentazione è evidentemente differente da quella per l'S-espressione $At1\ At2\ At1$ per la quale l'albero binario è composto da una catena, ma con frecce che ora puntano sempre in basso a destra.

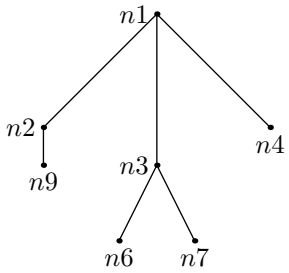
Oss. 10.1 *Alcune osservazioni rilevanti:*

- *Per come sono definite le liste, possono sempre essere indicate con delle S-espressioni (ne formano un sottoinsieme). Invece non tutto le S-espressioni possono essere scritte come liste, per esempio* $At1.At2$ *(senza* Nil*) in una lista possono occorrere come lettere.*
- *L'atomo* Nil *oltre che atomo è anche lista.*
- *Sia con le S-espressioni che con le liste non possono essere indicati degli "alberi" in memoria che "ricorrano su se stessi", come nell'esempio iniziale di organizzazione di memoria. Questi "alberi" non sono ammessi nel LISP, ma non possono essere generati con le sole notazioni predette (ne con le istruzioni elementari per le S-espressioni). Per gestire tali alberi impropri esistono delle istruzioni speciali che non vedremo.*
- *Tra gli atomi si considerano vari casi:*
  - *Sequenze alfanumeriche scelte dal programmatore (per indicare variabili, nomi di procedure, etc.). Queste sono organizzate in una tabella.*
  - *Costanti numeriche per vari numeri e diverse rappresentazioni.*
  - *Costanti logiche, per il vero (T) e il falso (talvolta si usa il* Nil*).*
  - *Parole chiave del linguaggio (in particolare LAMBDA, identificatori di funzioni elementari o primitive).*

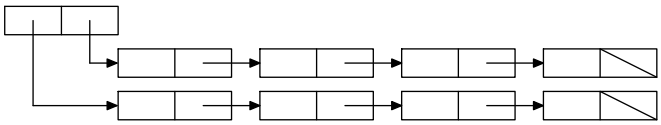
10.3 Uso delle S-espressioni

Le S-espressioni hanno due principali utilizzi: rappresentano strutture complesse di dati e l'organizzazione del programma sorgente sorgente.

a) Rappresentazione di strutture complesse di dati. Oltre agli alberi binari e alle parole si possono rappresentare alberi generali (per esempio gli alberi delle strategie del gioco degli scacchi che hanno nodi i cui discendenti determinati dalle mosse possibili dell'avversario variano nel progredire del gioco). Per fare ciò è possibile rappresentare i discendenti di un nodo con delle liste.



Ove $n1$ è la lista $n2, n3, n4$; $n2$ è la lista $n9$; $n3$ è la lista $n6, n7$ ed $n4$ è la lista () ovvero Nil. Si possono rappresentare con le liste (o con S-espressioni) matrici o tabelle a più indici. Per esempio una tabella a 2 indici 3×2 può essere rappresentata da:



Si possono rappresentare facilmente le strutture dati corrispondenti a dispositivi speciali di memorie (pile, code, etc.).

b) Rappresentazione dell'organizzazione del programma sorgente.

Esempio. 35 *La lista LAMBDA (lista1) (lista2) rappresenta la procedura corrispondente al λ-termine: $\lambda x_1 \cdots x_n.T$, ove (lista1) $= (x_1 \cdots x_n)$ e (lista2) $= T$. Cioè (lista1) individua i parametri di chiamata e (lista2) il corpo della procedura.*

Tali procedure sono sostanzialmente delle macro che vengono eseguite quando si applicano dei dati "corrispondenti a parametri di chiamata", vedremo poi in cosa consiste l'esecuzione.

Esempio. 36 *Mentre la lista precedente fornisce una macro parametrizzata (che conunque deve essere riscritta ogni volta che la si deve usare) la*

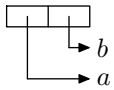
lista DEFINE (lista1) permette di associare a uno o più di tali macro dei nomi richiamabili (mediante lista1).

In generale tutto un programma sorgente scritto dal programmatore (o autogeneratosi) è una lista (LISP = LISt Processor).

10.4 Funzioni elementari

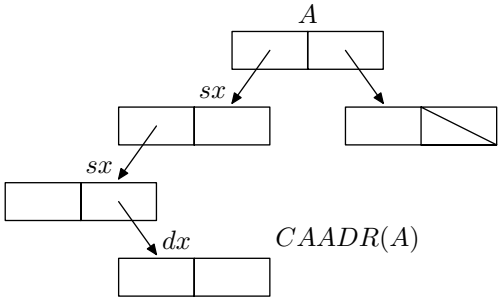
Il primo gruppo di funzioni elementari è quello delle istruzioni per gestire S-espressioni.

- CONS(A B) crea l'albero che ha come due discendenti delle S-espressioni A e B. Cioè aggiunge in memoria una cella di tipo:



  Ove a e b sono gli indirizzi delle (persistenti) radici di A e B.

- CAR e CDR fanno l'opposto: CAR(A) trova l'indirizzo del sottoalbero di sinistra di A, mentre CDR(A) trova quello di destra. Queste due istruzioni di etrazione ad un passo generano poi quelle a più passi, per esempio, CAADR(A) trova:



Oss. 10.2 *Le istruzioni elementari composte come le precedenti sono sostanzialmente parole binarie (Caadr corrisponde a* 0010*). Possiamo dire che le parole binarie non sono altro che gli indirizzi verso i nodi di un albero binario (abbastanza grande). La stessa cosa succede negli alberi unari (catene finite) che possiamo identificare coi numeri naturali. Per precisare con un esempio, tali alberi unari sono gli indirizzi per una (grande) memoria ad accesso diretto tradizionale. Per passare da un indirizzo ad un altro (precedente), cioè per passare dal vertice di un'albero unario ad un nodo precedente, usiamo numeri relativi negativi. Quindi tali istruzioni composte sembrano i corrispondenti dei numeri relativi tradizionali rispetto ad un ambiente binario (anziché unario).*

Nasce allora il problema di verificare tale intuizione e più in generale di vedere se esistano numeri "duo-ari" relativi in grado di trasformare alberi binari e possibilmente con un embrione di aritmentica[3].

10.4.1 Altre funzioni elementari

- Funzioni elementari aritmetiche: sommatoria, produttoria, esponenziali, etc.

- Funzioni elementari logiche: connettivi logici, predicati, etc.

10.5 Espressioni condizionali

LISP adotta, come nei linguaggi imperativi, espressioni condizionali, ma di tipo multiplo.

Esempio. 37 *COND(*$p1$, $e1$*)(*$p2$, $e2$*) significa che se il predicato* $p1$ *è vero si calcola* $e1$*, se* $p2$ *è vero si calcola* $e2$*.*

10.6 Istruzioni di controllo esecuzione

Normalmente il LISP esegue sequenzialmente le istruzioni (come negli altri linguaggi). Tuttavia vi sono situazioni in cui "non sa cosa fare". Per esempio se una procedura ha lo scopo di genrare altre procedure, quando viene eseguita la prima non si sa che fare della seconda. Si potrebbe, infatti, volerla eseguire oppure si potrebbe usarla, senza eseguirla subito, per costruire una terza procedura.

Allora il LISP ha la convenzione che normalmente la seconda procedura non venga eseguita. Per comandargli invece l'esecuzione occorre inserire un'istruzione apposita (EVAL).

In molti casi sussiste il problema inverso: si giunge ad una procedura (per esempio seguendo l'ordine sequenziale di esecuzione) che tuttavia *non* si deve eseguire al momento. A tale scopo esiste l'istruzione QUOTE che sospende l'esecuzione nel senso che quanto "quotato" resta invariato.

Esempio. 38 *La procedura (LAMBDA(X)(CAR X))(A.B) diventa CAR (A.B) = A. Invece la procedura (LAMBDA(X)(QUOTE X))(A.B) diventa X piché l'esecuzione ora consiste nel "non eseguire".*

Esiste anche una funzione TRACE che (almeno nella prima versione LISP) ha solo scopi diagnostici cioè serve a stampare l'andamento dell'esecuzione per "sbacare" il programma.

[3]Per quelli negativi, cioè parole binarie, conosciamo la catenazione che corrisponde in ambito unario alla somma. Ma ci sarà una somma per tutti gli ipotetici numeri "duari" relativi?

10.7 Ricorsione

Alcuni linguaggi di programmazione non consentono di programmare facilmente la ricorsione (quando si diffuse il LISP tutti gli altri non la consentivano). Se si vuole definire il fattoriale in un linguaggio non predisposto per la ricorsione, occorre organizzare la memoria affinchè ciò sia possibile: definire una pila con la quale realizzare i passi induttivi, etc. Invece con il LISP si può scrivere:

$$\text{Factorial } (n) = (\text{COND}((\text{ZEROP } n)1))$$

ZEROP n controlla che n sia 0, se si "restituisce" 1, altrimenti:

$$\text{Factorial } (n) = (\text{TIMES n}(\text{Factorial}(\text{SUB } 1\ n)))$$

TIMES n m moltiplica n per m, SUB n m sottrae n ad m.

Tali istruzioni possono quindi essere messe in una macro LAMBDA e con una definizione rese riutilizzabili.

Il motivo per cui il LISP non necessita di programmare la memoria è perchè esso valuta le funzioni (sia quelle predefinite che quelle create dal programmatore) semplicemente legando gli indirizzi dei parametri formali a quelli degli argomenti. Ciò è reso possibile dall'organizzazione della memoria virtuale LISP che è basata su alberi fatti con puntatori (indirizzi).