Sugli Approcci alla Teoria Astratta della Computabilità

Autore: Manzonetto Giulio

Mestre (Venezia - Italia), 30 ottobre 2002

Indice

Elenco delle figure

Abstract

Il problema di caratterizzare formalmente la classe delle funzioni effettivamente calcolabili è alla base della Teoria della Computabilità. In questa tesi verranno presentati i più importanti formalismi computazionali quali la teoria delle funzioni ricorsive parziali, il lambda calcolo e la logica combinatoria. Verranno inoltre analizzati ulteriori approcci assiomatici quali le Strutture Applicative Effettive (EAS), la Teoria delle Funzioni Ricorsive di Base (BRFT) e le Strutture Uniformemente Riflessive (URS). Tutti questi formalismi saranno confrontati e analizzati mettendone in evidenza analogie e differenze.

Capitolo 1

Introduzione

In questa tesi vengono esaminati alcuni fra i più importanti formalismi computazionali. La Teoria delle Funzioni Ricorsive Parziali (di Gödel e Kleene) è stata uno dei primi formalismi universali di calcolo. Essa individua la minima classe di funzioni contenente le proiezioni, il successore e lo zero chiusa rispetto alla composizione, alla minimalizzazione e alla ricorsione primitiva. Un altro formalismo di calcolo universale è il λ-calcolo non tipato. Il λ-calcolo, introdotto da Church intorno al 1930, è una teoria del concetto di funzione come regola di calcolo. In questa teoria senza tipi ciascun termine è, nel contempo, funzione e argomento. La sintassi del λ-calcolo è molto semplice: le funzioni sono rappresentate da espressioni, dette *λ-termini*, che si ottengono a partire da un insieme infinito di variabili $\{x, y, z, \ldots\}$ mediante applicazione ed astrazione funzionale.

I λ-termini sono contemporaneamente funzione e argomento, perciò ogni λ-termine può essere applicato anche a sè stesso. Per questa caratteristica tale formalismo rappresenta un prototipo dei linguaggi di programmazione funzionale, in cui è possibile studiare con gli strumenti formali del λ-calcolo le proprietà dei linguaggi di paradigma funzionale. Le computazioni nel λ-calcolo si basano sulla regola di β-conversione. L'applicazione di una funzione $\lambda x.M$ ad un argomento N viene valutata sostituendo l'argomento N al parametro formale x in M, e si scrive:

$$(\lambda x.M)N = M[x := N].$$

Una delle caratteristiche fondamentali del λ-calcolo che lo differenzia da tutti gli altri formalismi è la non distinzione fra programma, input-output e configurazione (o stato). Tutte queste cose sono sempre rappresentate da λ-termini liberando così questo linguaggio dal problema della codifica dei dati.

Un ulteriore formalismo che sarà trattato, sempre basato sulla non distinzione tra funzioni ed argomenti, è la Logica Combinatoria di Schönfinkel e Curry. Questo sistema si basa sull'idea di introdurre un nuovo simbolo ogni qual volta vi è la necessità di rappresentare una nuova funzione. Essi dimostrarono che qualsiasi funzione è esprimibile in termini di due soli combinatori $\mathbf{K}$ e $\mathbf{S}$.

Anche la sintassi della logica combinatoria è semplice. Le funzioni sono rappresentate da espressioni, dette *termini combinatori* o semplicemente *combinatori*, che si ottengono da $\mathbf{K}, \mathbf{S}$ e dalle variabili mediante l'applicazione. Le computazioni si basano sul calcolo equazionale con assiomi $\mathbf{K}xy = x$, $\mathbf{S}xyz = xz(yz)$. Un aspetto importante per cui il λ-calcolo differisce dalla Logica Combinatoria è che, in quest'ultima, non esiste il concetto di λ-astrazione. Nonostante ciò, si vedrà che il potere espressivo della Logica Combinatoria è pressochè il medesimo del λ-calcolo.

Nel capitolo finale della tesi verranno analizzate le Strutture Applicative Effettive (EAS), introdotte e studiate da Asperti e Ciabattoni. Una EAS è una famiglia Φ^n di funzioni:

$$\Phi^n : X \to \underbrace{(X^n \to X)}_{parziale} \quad n \in \mathbb{N}$$

con X insieme arbitrario e chiusa rispetto alla composizione, che contiene tutte le funzioni di proiezione, le funzioni S^m_n del teorema s-m-n della teoria classica della computabilità e contiene gli interpreti che rappresentano l'applicazione generica. Verrà dimostrata l'equivalenza fra le EAS e le algebre combinatorie parziali (PCA) che sono la versione parziale della Logica Combinatoria. PCA ed EAS sono una generalizzazione, rispettivamente, dell'algebra combinatoria parziale definita sui naturali e della teoria classica della computabilità. Con questa generalizzazione si intende effettuare un'operazione di astrazione su questi formalismi in modo tale che essi possano lavorare su un insieme arbitrario (che non sia necessariamente $\mathbb{N}$). Il pregio delle Strutture Applicative Effettive è quello di fungere da collegamento fra le PCA e teorie assiomatiche quali la Teoria delle Funzioni Ricorsive di Base e le Strutture Uniformemente Riflessive. Tutti questi formalismi saranno confrontati e analizzati mettendone in evidenza analogie e differenze.

Capitolo 2

Introduzione

Negli ultimi settant'anni molti ricercatori hanno indagato nel campo dell'*Informatica Teorica* che viene oggigiorno chiamato *Teoria della Computabilità*. Si sono altresì interrogati su quali problemi fossero risolvibili algoritmicamente cercando di dare una caratterizzazione matematica precisa all'idea intuitiva di *effettivamente computabile*. Fra gli scienziati più eminenti che hanno formulato una loro proposta in questo interessante settore scientifico ai margini fra la matematica e l'informatica ricordiamo Alonzo Church, Kurt Gödel, Stephen Kleene, Alan Turing, Emil Post, Charles Babbage, Robin Gandy e Reuben Goodstein. Dal lavoro di costoro e di altri sono nati svariati modelli e caratterizzazioni delle funzioni intuitivamente calcolabili. A seconda dei casi, può essere maggiormente vantaggioso l'utilizzo di un formalismo piuttosto che di un altro conformemente alle esigenze e al lavoro da svolgere ed è bene quindi sapere in cosa essi siano equivalenti e in cosa differiscano. In questa tesi considereremo alcune fra le teorie più interessanti cercando di metterne in luce le caratteristiche principali e le classi di funzioni rappresentabili in esse.

2.1 Teoria delle funzioni ricorsive

Per cominciare introdurremo la *teoria delle funzioni ricorsive parziali* di innegabile importanza storica (fu formalizzata da Gödel e Kleene 1936). Le funzioni ricorsive sono funzioni numeriche parziali del tipo

$f : \mathbb{N}^m \longrightarrow \mathbb{N}$. Daremo una definizione induttiva di tali funzioni introducendo parallelamente il concetto di *dimensione di una funzione ricorsiva* f che indicheremo con $Dim(f)$. Inoltre diremo che f è *definita* su x e scriveremo $f(x) \downarrow$ se la funzione f è definita su x, $f(x) \uparrow$ altrimenti.

Definizione 2.1.1 *La classe $\mathscr{R}$ delle funzioni ricorsive è una classe di funzioni numeriche parziali.*

$\mathscr{R}$ contiene le funzioni di base:

(i) proiezioni: $U_i^m(n_1, \ldots, n_m) = n_i$ *per ogni* $i \in \{1, \ldots, m\}$, $Dim(U_i^m) =_{def} m$;

(ii) successore: $S(n) = n + 1$, $Dim(S) =_{def} 1$;

(iii) zero: $Z(n) = 0$, $Dim(Z) =_{def} 1$;

La classe $\mathscr{R}$ è chiusa per minimalizzazione, composizione *e* ricorsione primitiva*:*

(i) minimalizzazione: sia $g \in \mathscr{R}$ *una funzione ricorsiva tale che* $Dim(g) = m + 1$. *Chiamiamo minimalizzazione di* g *la funzione ricorsiva* $M(g)$ *definita come segue:*

$$(M(g))(n_1, \ldots, n_m) = \begin{cases} \text{minimo } k \text{ tale che} & \text{se tale } k \text{ esiste} \\ g(k, n_1, \ldots, n_m) = 0 & \\ \text{ e } \forall i \ (0 \leq i < k) & \\ \Rightarrow g(i, n_1, \ldots, n_m) \downarrow & \\ & \\ \uparrow & \text{altrimenti.} \end{cases}$$

e, per definizione, $Dim(M(g)) =_{def} m$;

(ii) composizione: sia g *una funzione ricorsiva con* $Dim(g) = k$ *e* $h_1, \ldots, h_k$ *funzioni ricorsive tali che* $Dim(h_i) = m$ *per ogni* $i \in \{1, \ldots, k\}$ *chiamiamo composizione di* $g, h_1, \ldots, h_k$ *la funzione ricorsiva* $C(g, h_1, \ldots, h_k)$. *Definiamo induttivamente* $C(g, h_1, \ldots, h_k)$ *e il suo dominio come segue.*

Sia dato $\bar{x} = (n_1, \ldots, n_m)$*, allora:*

$$\bar{x} \in Dom(C(g, h_1, \ldots, h_k)) \ sse$$

$$(h_1(\bar{x})\downarrow, \ldots, h_k(\bar{x})\downarrow) \wedge (g(h_1(\bar{x}), \ldots, h_k(\bar{x}))\downarrow);$$

$$(C(g, h_1, \ldots, h_k))(\bar{x}) = \begin{cases} g(h_1(\bar{x}), \ldots, h_k(\bar{x})) & se\ \bar{x} \in Dom(C(g, h_1, \ldots, h_k)) \\ \uparrow & altrimenti. \end{cases}$$

Per definizione, poniamo $Dim(C(g, h_1, \ldots, h_k)) =_{def} m$*;*

(iii) ricorsione primitiva: siano g *e* h *funzioni ricorsive tali che* $Dim(g) = m$ *e* $Dim(h) = m + 2$*. Possiamo definire induttivamente la funzione ricorsiva* f *e il suo dominio come segue:*

$$(0, n_1, \ldots, n_m) \in Dom(f) \ sse\ g(n_1, \ldots, n_m)\downarrow$$

$$(n + 1, n_1, \ldots, n_m) \in Dom(f)$$

$$sse$$

$$((n, n_1, \ldots, n_m) \in Dom(f)) \wedge h(f(n, n_1, \ldots, n_m), n, n_1, \ldots, n_m)\downarrow$$

$$f(n, n_1, \ldots, n_m) = \begin{cases} g(n_1, \ldots, n_m) & se\ (n = 0) \wedge \\ & ((0, n_1, \ldots, n_m) \in Dom(f)); \\ & \\ h(f(n-1, n_1, \ldots, n_m), & se\ (n \neq 0) \wedge \\ n-1, n_1, \ldots, n_m) & ((n, n_1, \ldots, n_m) \in Dom(f)); \\ & \\ \uparrow & altrimenti. \end{cases}$$

Per definizione, poniamo $Dim(f) =_{def} m + 1$*.*

Consideriamo ora, a titolo d'esempio, la funzione che calcola il fattoriale di un numero e traduciamola nel formalismo appena introdotto delle funzioni ricorsive. Ricordiamo che il fattoriale, matematicamente, è definito come segue:

$$fatt : \mathbb{N} \longrightarrow \mathbb{N}$$

$$fatt(n) = \begin{cases} 1 & \text{se } (n = 0) \\ mul(fatt(n-1), n) & \text{altrimenti.} \end{cases}$$

$$mul : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$$

$$mul(x, y) = \begin{cases} 0 & \text{se } (x = 0) \\ sum(mul(x-1, y), y) & \text{altrimenti.} \end{cases}$$

$$sum : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$$

$$sum(x, y) = \begin{cases} y & \text{se } (x = 0) \\ sum(x-1, y) + 1 & \text{altrimenti.} \end{cases}$$

Nella teoria delle funzioni ricorsive parziali è possibile scrivere il fattoriale, applicando le precedenti definizioni, nel seguente modo:

$$Dim(fatt) = 1;\ \ Dom(fatt) = \mathbb{N};$$

$$fatt(n) = \begin{cases} g_1 & \text{se } (n = 0) \wedge \\ & ((0) \in Dom(fatt)); \\ & \\ h_1(fatt(n-1), n-1) & \text{se } (n \neq 0) \wedge \\ & (n \in Dom(fatt)); \\ & \\ \uparrow & \text{altrimenti.} \end{cases}$$

con:

$$Dim(g_1) = 0;\ \ g_1 =_{def} C(S, M(Z));$$

$$Dim(h_1) = 2;\ \ h_1 =_{def} C(mul, U_1^2, C(S, U_2^2)).$$

$$Dim(mul) = 2; Dom(mul) = \mathbb{N} \times \mathbb{N};$$

$$mul(x, y) = \begin{cases} g_2(y) & \text{se } (x = 0) \wedge \\ & ((0, y) \in Dom(mul)); \\ & \\ h_2(mul(x-1, y), x-1, y) & \text{se } (x \neq 0) \wedge \\ & ((x, y) \in Dom(mul)); \\ & \\ \uparrow & \text{altrimenti.} \end{cases}$$

con:

$$Dim(g_2) = 1;\ g_2 =_{def} Z;$$

$$Dim(h_2) = 3;\ h_2 =_{def} C(sum, U_1^3, U_3^3).$$

$$Dim(sum) = 2;\ Dom(sum) = \mathbb{N} \times \mathbb{N};$$

$$sum(x, y) = \begin{cases} g_3(y) & \text{se } (x = 0) \wedge \\ & ((0, y) \in Dom(sum)); \\ & \\ h_3(mul(x-1, y), x-1, y) & \text{se } (x \neq 0) \wedge \\ & ((x, y) \in Dom(sum)); \\ & \\ \uparrow & \text{altrimenti.} \end{cases}$$

con:

$$Dim(g_3) = 1;\ g_3 =_{def} U_1^1;$$

$$Dim(h_3) = 3;\ h_3 =_{def} C(S, U_1^3).$$

Vedremo ora come una funzione ricorsiva individui in modo naturale una funzione intesa in senso insiemistico. Più precisamente:

Osservazione 2.1.2 *Sia data una $F \in \mathscr{R}$, con $dim(F) = k$. Si dice* funzione indotta *da F la funzione $f : \Omega \longrightarrow \mathbb{N}$ tale che:*

(i) $\Omega = \{\omega \in \mathbb{N}^k | F \downarrow \omega\}$;

(ii) per ogni $\omega \in \Omega$ è $f(\omega) = F(\omega)$.

Osservazione 2.1.3 *Consideriamo un sottoinsieme Ω di $\mathbb{N}^k$ ed una funzione $f : \Omega \longrightarrow \mathbb{N}$. Si dice che $F \in \mathscr{R}$* rappresenta *f sse la funzione indotta da F coincide con f.*

Osservazione 2.1.4 *Si dice che f è* rappresentabile *sse esiste $F \in \mathscr{R}$ che rappresenta f.*

Si può mostrare che non tutte le funzioni sono rappresentabili. Come possiamo sapere che una funzione intuitivamente calcolabile è effettivamente rappresentabile nella teoria delle funzioni ricorsive? Giungono in nostro soccorso i risultati di Turing e Church che ci accingiamo ad esaminare nella prossima sezione.

2.2 Macchina di Turing e Tesi di Church

Alan Turing nel 1936 definisce un automa astratto chiamato *macchina di Turing* basato sull'analisi di un soggetto umano che implementa un algoritmo usando carta (un nastro infinito suddiviso in quadretti) e penna. Ad ogni istante l'attenzione può essere posta su un unico quadretto del nastro. Tramite tale approccio l'esecuzione di un algoritmo viene visto come una successione di azioni molto semplici del tipo:

(i) scrivere o cancellare un simbolo;

(ii) spostare l'attenzione da un quadretto all'altro sul nastro di carta.

In ogni momento l'algoritmo specifica che azione deve compiere il soggetto umano a seconda dei simboli scritti sul nastro e dello stato mentale in cui si trova l'esecutore.

Formalmente una Macchina di Turing (MdT) M è una quintupla definita come segue:

$$M = (Q, \Gamma, \delta, q_0, B),$$

in cui

Q	è un insieme finito di *stati*;
Γ	è un insieme finito di simboli detto alfabeto;
δ	è una funzione $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{D, S\}$; ($\delta$ può anche essere indefinita su alcuni input, in tal caso la macchina di Turing termina l'esecuzione del programma);
q_0	appartenente a Q è lo stato iniziale;
B	è un simbolo appartenente a Γ corrispondente al carattere vuoto.

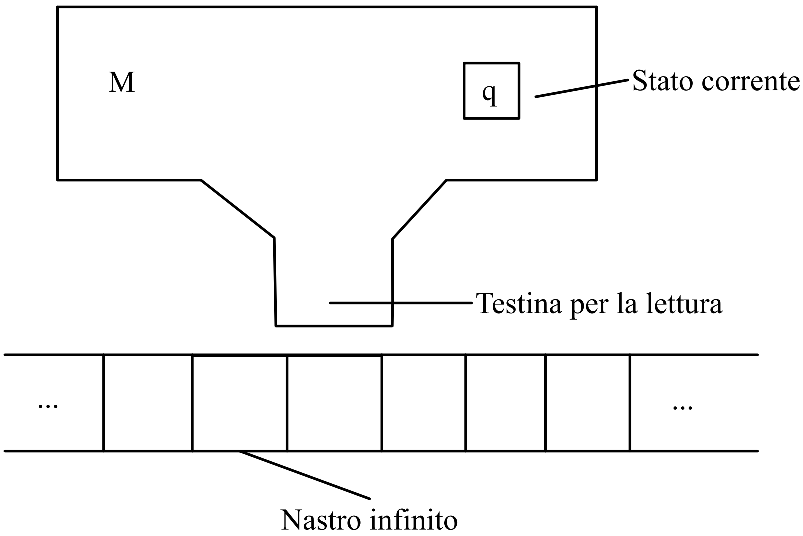

Figura 2.1: Macchina di Turing

All'inizio dell'esecuzione di un programma la Macchina di Turing si trova in stato q_0, sul nastro è presente la stringa $\alpha \in \Gamma^*$[1] e la testina della macchina è posizionata sul simbolo più a sinistra di α. Nel caso in cui α sia la stringa vuota la testina leggerà B. Un programma scritto per una Macchina di Turing $M = (Q, \Gamma, \delta, q_0, B)$ è un insieme finito di quintuple del tipo:

$$< i, q_a, q_b, o, d >$$

[1]con Γ^* si indica la chiusura di Kleene dell'insieme Γ, cioè l'insieme di tutte le stringhe formate tramite la giustapposizione di simboli contenuti in Γ, compresa la stringa vuota.

che sono le istruzioni del programma in cui

$i \in \Gamma$	è il simbolo letto dalla testina sul nastro(input);
$q_a \in Q$	è lo stato in cui si trova la MdT;
$q_b \in Q$	è lo stato in cui transita la MdT una volta eseguita l'istruzione;
$o \in \Gamma$	è il simbolo scritto dalla testina sul nastro(output);
$d \in \{D, S\}$	è la direzione (D = destra, S = sinistra) in cui si muoverà la testina sul nastro una volta terminata l'istruzione.

Ogni istruzione del tipo $< i, q_a, q_b, o, d >$ corrisponde a definire $\delta(i, q_a) = (q_b, o, d)$. Se durante l'esecuzione di un programma la Macchina di Turing si trova in stato q_i e riceve in input il simbolo x e δ non è definita sulla coppia (q_i, x) il programma termina l'esecuzione.

Osservazione 2.2.1 *Qualsiasi funzione possa essere implementata tramite una macchina di Turing si dice Turing-computabile.*

Denotiamo con $\mathscr{C}$ la classe delle funzioni Turing-computabili.

Tesi 2.2.2 *Tesi di Church.*
La classe delle funzioni numeriche calcolabili con un algoritmo informale coincide esattamente con la classe $\mathscr{C}$ delle funzioni Turing-computabili.

Tale tesi non può essere dimostrata in quanto non si può definire cosa sia calcolabile con un algoritmo informale ma, per la ragionevolezza del suo enunciato derivata dall'esperienza pratica, la riterremo egualmente vera.

Osservazione 2.2.3 *Si può dimostrare, come viene fatto in [Cutland, 1980] che la classe $\mathscr{C}$ delle funzioni Turing-computabili coincide con la classe $\mathscr{R}$ delle funzioni ricorsive.*

La tesi di Church e l'osservazione 2.2.3 sono assai importanti per i nostri obiettivi in quanto sottolineano il fatto che, da un lato esiste un limite massimo alle funzioni che possiamo scrivere con i formalismi fino ad oggi introdotti, dall'altro essi sono abbastanza potenti da consentirci di rappresentare qualsiasi funzione intuitivamente calcolabile.

Capitolo 3

λ-calcolo non tipato

In questo capitolo introdurremo i concetti fondamentali riguardanti il λ-calcolo, indagheremo sul suo potere espressivo confrontandolo con quello della Teoria delle funzioni ricorsive e definiremo cos'è una λ-teoria.

3.1 Introduzione al λ-calcolo

Il λ-calcolo è una teoria formale non tipata che fu proposta da Alonzo Church nel 1932 come teoria fondazionale per la matematica e che diventò di rilevante interesse nel campo dell'informatica teorica e della logica. Il sistema iniziale di Church, che prevedeva anche elementi di logica, era più generale di quello conosciuto attualmente. Sfortunatamente Kleene e Rosser, due studenti di Church, nel 1935 dimostrarono l'inconsistenza[1] di tale teoria. L'anno successivo Church, continuando i suoi studi coadiuvato dai suoi due allievi, riuscì ad isolarne un sottosistema consistente che è oggi conosciuto come λ-calcolo.

Il λ-calcolo è una teoria orientata alle funzioni ed è *non estensionale* cioè λ-termini distinti possono avere lo stesso comportamento funzionale. Uno degli aspetti che lo rende interessante è il fatto che ciascun termine è, nel contempo, funzione e argomento senza distinzione alcuna; ciò per-

[1] Una teoria è detta *inconsistente* se tutti i termini vengono eguagliati dagli assiomi vigenti.

mette di far lavorare un programma(inteso come funzione) su un altro in modo automatico, senza cioè essere costretti a cercare, come nella teoria classica della computabilità, una funzione effettiva di codifica definita dall'insieme dei programmi a quello degli argomenti.

Numerose proprietà del λ-calcolo non verranno trattate in questo elaborato in quanto esulano dai nostri scopi; si consiglia a chi fosse interessato ad approfondire tali argomenti la consultazione di [Barendregt, 1984] e [Sòrensen and Urzyczyn, 1998].

3.2 Interpretazione Informale del λ-calcolo

Prima di fornire una corretta definizione formale del λ-calcolo conviene spiegare in maniera meno matematica ma sicuramente più intuitiva le idee che stanno alla base di questo formalismo computazionale per far sì che anche chi si avvicina per la prima volta a tali concetti riesca a comprenderli efficacemente. Partiamo dunque dalla nozione di funzione ricordando come viene solitamente espressa nei formalismi matematici. Consideriamo una espressione generica:

$$E$$

che, auspicabilmente, conterrà delle occorrenze della variabile x. Per poter costruire una funzione basandoci sulla nostra espressione E siamo soliti introdurre un nuovo simbolo, ad esempio f, e porre:

$$f(x) = E.$$

Il nuovo simbolo f è il nome della funzione.

Nel λ-calcolo la stessa funzione si scrive in tal modo:

$$\lambda x.E$$

in cui λ è l'operatore di astrazione funzionale ed è lo strumento che permette di rappresentare le funzioni di arietà uno, ossia dipendenti da un solo parametro. Possiamo calcolare il valore che assume $\lambda x.E$ su un argomento a sostituendo ogni occorrenza della variabile x in E con a. Tale operazione è svolta da $\rightarrow_\beta$.

Si consideri l'espressione aritmetica:

$$x^2 + x + 1$$

e si costruisca la funzione:

$$\lambda x.x^2 + x + 1$$

come visto in precedenza.

L'applicazione di un parametro, quale potrebbe essere 4, a $\lambda x.x^2{+}x{+}1$ produrrà le seguenti computazioni:

$$(\lambda x.x^2 + x + 1)4 \rightarrow_\beta x^2 + x + 1[x := 4]^2 \;=\; 4^2 + 4 + 1 = 21.$$

La rappresentazione delle funzioni di arietà maggiore di uno in λ-calcolo è consentita da una proprietà dimostrata da $Schönfinkel$ in [Schonfinkel, 1924] secondo la quale qualsiasi funzione a più variabili può essere ottenuta applicando sequenzialmente più funzioni unarie.

Consideriamo una funzione a due argomenti come può essere $f(x, y)$ vediamo in che modo è possibile scomporla in due funzioni di arietà uno che chiameremo F e F_x definite in tal modo:

$$F_x \equiv (\lambda y.f(x, y)) \quad F \equiv \lambda x.F_x$$

$$((F\ x)\ y) = (F_x\ y) = f(x, y).$$

3.3 Pre-termini

Gli oggetti di studio nel λ-calcolo sono i λ-*termini* ma, per poterli definire correttamente, è conveniente introdurre in primo luogo la nozione di *pre-termine*.

Definizione 3.3.1 *Siano dati:*

$$V = \{v_0, v_1, \ldots\}$$

[2]con $x^2 + x + 1[x := 4]$ si intende l'espressione $x^2 + x + 1$ in cui si sostituisce 4 ad ogni occorrenza di x.

un insieme che denota un alfabeto infinito di variabili, il simbolo λ *detto* operatore di astrazione*, il simbolo* . *detto* operatore di applicazione *e i simboli (e) che consentono la* parentesizzazione*. Viene chiamato* Λ^- *l'insieme dei* pre-termini *cioè l'insieme di tutte le stringhe definite tramite la seguente grammatica espressa in BNF*[3]*:*

$$\Lambda^- ::= V|(\Lambda^-\Lambda^-)|(\lambda V\Lambda^-)$$

Le stringhe seguenti sono pre-termini.

(i) $(((v_0\ v_1)\ v_2)\ v_3) \in \Lambda^-$;

(ii) $(\lambda v_0(\lambda v_1\ v_0)) \in \Lambda^-$;

(iii) $(\lambda v_0(v_0\ v_1)) \in \Lambda^-$;

(iv) $(\lambda v_0(v_0\ v_1))(\lambda v_2(v_2\ v_3)) \in \Lambda^-$.

Per denotare elementi arbitrari di Λ^- si è soliti utilizzare lettere maiuscole quali K, L, M, N, O, P, Q, R. Verranno riservate, invece, lettere minuscole quali x, y, z, per denotare elementi arbitrari di V.

Terminologia 3.3.2 *Per convenzione:*

(i) Un pre-termine della forma x con $x \in V$ si dice variabile*;*

(ii) Un pre-termine della forma $(\lambda x\ M)$ si dice λ-astrazione *su x;*

(iii) Un pre-termine della forma $(M\ N)$ si dice applicazione *di M a N.*

La parentesizzazione completa è spesso sconveniente per problemi di leggibilità. Si preferisce introdurre, quindi, una convenzione per l'eliminazione delle parentesi ridondanti. In via informale conviene pensare che l'operatore di applicazione associa verso sinistra mentre la λ-astrazione associa quanto più a destra è consentito.

[3]Ovvero Backus Naur Form, una notazione introdotta da J.Backus agli inizi degli anni '60. Naur era il curatore di una rivista, *l'Algol 60 Report*, in cui - per la prima volta - comparve la sintassi di un linguaggio di programmazione formalizzata in modo preciso.

Notazione 3.3.3 *Siano utilizzate le seguenti abbreviazioni:*

(i) $(M\ P\ Q)$ *al posto di* $((M\ P)\ Q)$ *;*

(ii) $(\lambda x\ \lambda y\ M)$ *al posto di* $(\lambda x\ (\lambda y\ M))$ *;*

(iii) $\lambda x\ M\ N$ *al posto di* $\lambda x\ (M\ N)$ *;*

(iv) $(M\ \lambda x\ N)$ *al posto di* $(M\ (\lambda x\ N))$ *.*

Possiamo dunque omettere tali parentesi.

A questo punto è utile introdurre un altra piccola variazione nella notazione adottata.

Notazione 3.3.4 *Scriveremo:*

$$\lambda x_1 \ldots x_n.M \quad \text{in luogo di} \quad \lambda x_1 \ldots \lambda x_n\ M\ .$$

Come caso specifico di questa regola generale con $\lambda x.M$ *si intende* $\lambda x\ M$*.*

Con questi nuovi accorgimenti sintattici possiamo riscrivere i pretermini precedentemente elencati come segue:

(i) $v_0\ v_1\ v_2\ v_3\ \in \Lambda^-$;

(ii) $\lambda v_0 v_1.v_0\ \in \Lambda^-$;

(iii) $(\lambda v_0.v_0\ v_1)\ \in \Lambda^-$;

(iv) $(\lambda v_0.v_0\ v_1)\ \lambda v_2.v_2\ v_3 \in \Lambda^-$.

Spesso si continuerà a preferire $(\lambda v_0.v_0\ v_1)(\lambda v_2.v_2\ v_3)$ in luogo di $(\lambda v_0.v_0\ v_1)\ \lambda v_2.v_2\ v_3$ per consentire una maggior leggibilità.

Definizione 3.3.5 *Sia* $M \in \Lambda^-$*, allora si definisce ricorsivamente l'insieme* $FV(M)$ *delle* variabili libere *di* M *come segue:*

$$\begin{array}{lcl} FV(x) & = & \{x\}; \\ FV(\lambda x.P) & = & FV(P)\backslash\{x\}; \\ FV(P\ Q) & = & FV(P) \cup FV(Q). \end{array}$$

Un pre-termine $M \in \Lambda^-$ viene detto *chiuso* se $FV(M) = \{\}$. Forniamo ora la nozione di *sostituzione* che ci sarà utile, come abbiamo visto nel paragrafo 3.2, per svolgere le computazioni quando applichiamo a una funzione del λ-calcolo un argomento.

Definizione 3.3.6 *Siano M, $N \in \Lambda^-$ e $x \in V$ allora si dice* sostituzione *di x a N in M e si scrive $M[x := N] \in \Lambda^-$ l'operazione così definita:*

$$
\begin{array}{lcll}
x[x := N] & = & N; & \\
y[x := N] & = & y & \textit{con } x \neq y; \\
(M\ P)[x := N] & = & M[x := N]\ P[x := N]; & \\
(\lambda x.M)[x := N] & = & \lambda x.M; & \\
(\lambda y.M)[x := N] & = & \lambda y.M[x := N] & \textit{se } y \notin FV(N) \textit{ o } x \notin FV(M); \\
(\lambda y.M)[x := N] & = & \lambda z.M[y := z][x := N] & \textit{se } y \in FV(N) \textit{ e } x \in FV(M)
\end{array}
$$

con $z \in \{v_i \in V |\ v_i \notin FV(M) \cup FV(N)\}$.

In quest'ultima definizione potrebbe non risultare evidente perché la regola da applicare nel caso di un λ-termine del tipo $(\lambda y.M)[x := N]$ dipenda dall'appartenenza di x e y rispettivamente agli insiemi delle variabili libere di N e M. Forniamo quindi un esempio significativo delle problematiche che si potrebbero riscontrare se tale distinzione non venisse effettuata. Si consideri il λ-termine $(\lambda y.xyy)[x := yy]$. Se provassimo ad applicare la regola $(\lambda y.M)[x := N] = \lambda y.M[x := N]$ anche in questo caso (si noti che $y \in FV(N)$) si otterrebbe:

$$(\lambda y.xyy)[x := yy] = \lambda y.xyy[x := yy] = \lambda y.yyyy$$

Questo risultato è errato in quanto la variabile y che occorreva libera in N diventa legata dopo la sostituzione. La scorrettezza di tale sostituzione si può anche desumere dal fatto che il λ-termine $(\lambda y.xyy)$ ha lo stesso comportamento funzionale di $(\lambda z.xzz)$ quindi $(\lambda z.xzz)[x := yy]$ dovrebbe dare un risultato che ha, a sua volta, lo stesso comportamento funzionale di $(\lambda y.xyy)[x := yy]$. Invece

$$(\lambda z.xzz)[x := yy] = \lambda z.yyzz$$

che è il risultato corretto ed è stato ottenuto rinominando, prima di eseguire la sostituzione, la variabile y in $z \notin FV(N)$ che poi è l'operazione eseguita dall'ultima regola della definizione 3.3.6. Problemi analoghi si verificano anche se $x \in FV(M)$.

Come abbiamo visto nel paragrafo 3.2, i λ-pretermini rappresentano funzioni. Il risultato che otteniamo applicando un λ-termine ad un argomento non dipende dal nome delle variabili legate in esso contenute. È quindi necessario introdurre una relazione di congruenza che eguagli i termini che differiscono fra loro solo per il nome delle variabili legate.

Definizione 3.3.7 *Sia detta* α-equivalenza *o* α-congruenza *e si scriva* $=_\alpha$ *la più piccola relazione su* Λ^-*, tale che:*

$$\begin{array}{ll} M =_\alpha M & \textit{per ogni } M \in \Lambda^-; \\ \lambda x.M =_\alpha \lambda y.M[x := y] & \textit{se } y \notin FV(M), \end{array}$$

ed inoltre sia chiusa rispetto alle seguenti regole:

$$\begin{array}{lll} M =_\alpha M' & \Rightarrow & \forall x \in V \ : \quad \lambda x.M =_\alpha \lambda x.M'; \\ M =_\alpha M' & \Rightarrow & \forall Q \in \Lambda^- \ : \quad M\ Q =_\alpha M'\ Q; \\ M =_\alpha M' & \Rightarrow & \forall Q \in \Lambda^- \ : \quad Q\ M =_\alpha Q\ M'; \\ M =_\alpha M' & \Rightarrow & M' =_\alpha M; \\ M =_\alpha M' \ \&\ M' =_\alpha M'' & \Rightarrow & M =_\alpha M''. \end{array}$$

Osservazione 3.3.8 *La relazione* $=_\alpha$ *di* α*-equivalenza è decidibile cioè la* funzione caratteristica[4] *di tale relazione è rappresentabile tramite il formalismo delle funzioni ricorsive come definite in 2.1.1 a pagina 8.*

Definizione 3.3.9 *Si definisce* rinomina delle variabili legate *di* M*, con* $M \equiv \lambda x.N$[5]*, l'operazione* ρ *attraverso la quale si sostituisce ogni occorrenza in* M *della variabile legata* x *con una qualche variabile* z *che non sia già presente in* M*, cioè,* $\rho(x, M) = \lambda z.M[x := z]$ *con* $z \notin FV(M)$.

Siano considerati due termini arbitrari $M,\ N \ \in \Lambda^-$. È semplice verificare che M è α-equivalente ad N sse esiste un numero finito di operazioni di rinomina delle variabili legate di M che lo rende sintatticamente uguale a N.

[4]Per funzione caratteristica di una relazione binaria si intende la funzione f(x,y) che vale 1 se x è in relazione con y, 0 altrimenti.

[5]Si noti che con $\equiv$ si intende l'eguaglianza sintattica.

3.4 λ-termini

In questa sezione introduciamo i λ-termini che sono gli effettivi oggetti di studio del λ-calcolo. L'idea intuitiva è quella di considerare equivalenti tutti i λ-pretermini che differiscono solo per il nome delle variabili.

Definizione 3.4.1 *Definiamo per ogni $M \in \Lambda^-$ la* classe di equivalenza $[M]_\alpha$ *in tal modo:*

$$[M]_\alpha = \{N \in \Lambda^- | M =_\alpha N\}$$

Allora ci è possibile definire l'insieme Λ di λ-termini come:

$$\Lambda \quad = \quad \Lambda^- / =_\alpha \quad = \quad \{[M]_\alpha | M \in \Lambda^-\}$$

Per brevità, nel seguito della trattazione, scriveremo M in luogo di $[M]_\alpha$ eccetto quando verrà esplicitamente dichiarato il contrario. Ecco degli esempi di λ-termini.

(i) $\mathbf{K} \equiv \lambda xy.x \ \in \Lambda$;

(ii) $\mathbf{S} \equiv \lambda xyz.xz(yz) \ \in \Lambda$;

(iii) $\mathbf{I} \equiv \lambda x.x \ \in \Lambda$;

(iv) $\mathbf{1} \equiv \lambda xy.xy \in \Lambda$;

(v) $\mathbf{Y} \equiv \lambda f.(\lambda x.f(x\ x))(\lambda x.f(x\ x)) \ \in \Lambda$;

(vi) $\omega \equiv \lambda x.xx \in \Lambda$;

(vii) $\Omega \equiv \omega\omega \equiv (\lambda x.xx)(\lambda x.xx) \in \Lambda$.

Allo stesso modo di come abbiamo fatto per i pre-termini diamo le nozioni di variabili libere, variabili legate e chiusura di un termine anche per i λ-termini.

Osservazione 3.4.2 *Si noti che:*

(i) dati $M, N \in \Lambda$, allora $M =_\alpha N \Rightarrow FV(M) = FV(N)$;

(ii) I λ-termini ereditano le nozioni di insieme delle varibili libere e occorrenza di una variabile libera dai λ-pretermini.

Sia $M \in \Lambda$, allora una variabile x che occorre in M si dice *vincolata* o *legata* se $x \notin FV(M)$. Un λ-termine $M \in \Lambda$ viene detto *chiuso* se $FV(M) = \{\}$.

Definizione 3.4.3 *Sia $M \in \Lambda$, allora si definisce induttivamente l'insieme Sub(M) come segue:*

$$\begin{array}{l} Sub(x) = \{x\} \\ Sub(\lambda x.P) = \{\lambda x.P\} \ \cup \ Sub(P) \\ Sub(P\ Q) = \{P\ Q\} \ \cup \ Sub(P) \ \cup \ Sub(Q) \end{array}$$

Siano $M,\ N\ \in \Lambda$ allora si è soliti dire che M è *sottotermine* di N se $M \in Sub(N)$.

Osservazione 3.4.4 *Anche la nozione di sostituzione definita per i λ-pretermini viene ereditata dai λ-termini con l'accorgimento che se $M, N, M', N' \in \Lambda$ e $M =_\alpha M', N =_\alpha N'$, allora $M[x := N] =_\alpha M'[x := N']$.*

Forniamo di seguito alcuni esempi.

(i) $(\lambda x.x\ y)[y := z] = (\lambda x.x\ z)$;

(ii) $(x\ (y\ z))[y := \lambda k.k] = (x\ ((\lambda k.k)\ z))$.

3.5 Riduzione

Nel seguito di questo paragrafo introdurremo il concetto di *riduzione* sui λ-termini. L'operatore di riduzione è quello che permette di svolgere le computazioni quando viene applicato un argomento a un λ-termine.

Definizione 3.5.1 *Sia $\rightarrow_\beta$ la più piccola relazione su Λ tale che valga:*

$$(\lambda x.M)\ N \rightarrow_\beta M[x := N],$$

e che sia chiusa rispetto le seguenti regole:

$$\begin{array}{lcll} M \to_\beta M' & \Rightarrow & \forall x \in V: & \lambda x.M \to_\beta \lambda x.M' \\ M \to_\beta M' & \Rightarrow & \forall N \in \Lambda: & M\ N \to_\beta M'\ N \\ M \to_\beta M' & \Rightarrow & \forall N \in \Lambda: & N\ M \to_\beta N\ M' \end{array}$$

allora si dice che un termine della forma $(\lambda x.M)\ N$ *è un* β-redex *mentre* $M[x := N]$ *viene chiamato* β-contractum.

Un termine M è detto in *forma β-normale* (β-NF) se non esiste alcun termine N tale che $M \to_\beta N$ cioè se M non contiene β-redex.

Proposizione 3.5.2 *Esistono termini che non hanno una forma β-normale in quanto contengono β-redex il cui contractum contiene sempre un β-redex. In questo caso la riduzione continua all'infinito.*

Consideriamo questo esempio. Sia:

$$\Omega \equiv (\lambda x.x\ x)(\lambda x.x\ x).$$

Se si prova a svolgere la riduzione contenuta in Ω si ottiene :

$$\Omega \equiv (\lambda x.x\ x)(\lambda x.x\ x) \to_\beta x\ x[x := \lambda x.x\ x] \equiv (\lambda x.x\ x)(\lambda x.x\ x) \equiv \Omega$$

dunque $\Omega \to_\beta \Omega \to_\beta \ldots \to_\beta \Omega$. Tale termine, dunque, non solo non è in forma β-normale poichè contiene un β-redex ma addirittura non possiede una forma β-normale.

Consideriamo ora il termine $\mathbf{KI}\Omega$, esso contiene due redex, Ω e $\mathbf{KI}$, e a seconda di quello che si sceglie di ridurre si otterranno risultati diversi. Proviamo, ad esempio a ridurre il primo:

$$\mathbf{KI}\Omega \to_\beta \mathbf{KI}\Omega \to_\beta \ldots \to_\beta \mathbf{KI}\Omega$$

risulta evidente che le computazioni si protrarranno all'infinito. Vediamo, dunque, cosa accade se riduciamo il secondo:

$$\mathbf{KI}\Omega \to_\beta \mathbf{I}$$

che è una β-NF.

Per indicare che da un λ-termine M si può giungere ad un λ-termine N tramite una o più applicazioni (ma comunque in numero finito) di $\to_\beta$ si può usare la relazione $\twoheadrightarrow_\beta$ che viene formalmente definita come segue.

Definizione 3.5.3 *La relazione* $\twoheadrightarrow_\beta$ *viene chiamata* β-riduzione transitiva *altresì detta* β-riduzione a più passi *ed è la più piccola chiusura riflessiva e transitiva di* $\rightarrow_\beta$ *cioè* $\twoheadrightarrow_\beta$ *è la più piccola relazione chiusa rispetto le seguenti regole:*

$$\begin{array}{lcl} M \rightarrow_\beta M' & \Rightarrow & M \twoheadrightarrow_\beta M'; \\ M \twoheadrightarrow_\beta M' \ \& \ M' \twoheadrightarrow_\beta M'' & \Rightarrow & M \twoheadrightarrow_\beta M''; \\ M \twoheadrightarrow_\beta M. & & \end{array}$$

Un termine $M \in \Lambda$:

(i) si dice *β-normalizzabile* se è possibile ridurlo a una forma β-normale, cioè se esiste una β-NF N tale che $M \twoheadrightarrow_\beta N$;

(ii) si dice *fortemente β-normalizzabile* se, da esso, non è possibile generare una riduzione infinita;

(iii) si dice *β-infinito* se non è fortemente β-normalizzabile.

Definizione 3.5.4 *La relazione* $=_\beta$ *viene chiamata* β-equivalenza *o* β-congruenza *ed è la più piccola chiusura riflessiva, transitiva e simmetrica di* $\rightarrow_\beta$*, cioè* $=_\beta$ *è la più piccola relazione chiusa rispetto le seguenti regole:*

$$\begin{array}{lcl} M \rightarrow_\beta M' & \Rightarrow & M =_\beta M'; \\ M =_\beta M' \ \& \ M' =_\beta M'' & \Rightarrow & M =_\beta M''; \\ M =_\beta M; & & \\ M =_\beta M' & \Rightarrow & M' =_\beta M. \end{array}$$

Ecco alcuni esempi:

(i) $(\lambda x.x\ y)(\lambda z.z) \rightarrow_\beta (x\ y)[x := (\lambda z.z)] \equiv (\lambda z.z)y \rightarrow_\beta z[z := y] \equiv y$;

(ii) $(\lambda x.x\ x)(\lambda y.y) \twoheadrightarrow_\beta (\lambda z.z)$;

(iii) $(\lambda x.x\ x)\ y =_\beta y\ ((\lambda z.z)\ y)$.

Definizione 3.5.5 *Si dice che una relazione binaria R fra λ-termini gode della* proprietà del diamante *se esiste $Z \in \Lambda$ tale che:*

$$(M \to_R N_1)\&(M \to_R N_2) \Rightarrow (N_1 \to_R Z)\&(N_2 \to_R Z)$$

Notazione 3.5.6 *Sia data una relazione di riduzione R. Allora la sua chiusura riflessiva e transitiva si indica con $\twoheadrightarrow_R$.*

Sia data una relazione di riduzione R. Se $\twoheadrightarrow_R$ soddisfa la proprietà del diamante di dice che R gode della *proprietà di Church-Rosser*.

Teorema 3.5.7 *Sia R una relazione che gode della proprietà di Church-Rosser, allora esite uno Z tale che:*

$$M =_R N \Rightarrow (M \twoheadrightarrow_R Z)\&(N \twoheadrightarrow_R Z)$$

Proposizione 3.5.8 *Se una relazione binaria R gode della proprietà di Church-Rosser allora valgono le seguenti proprietà:*

(i) ogni λ-termine ammette al massimo una forma R-normale[6]*;*

(ii) se un λ-termine è R-normalizzabile allora esiste almeno un cammino di riduzione che arriva a tale forma.

Teorema 3.5.9 (Church Rosser)

(i) $\twoheadrightarrow_\beta$ gode della proprietà di Church-Rosser

(ii) $M =_\beta N \Rightarrow \exists Z[(M \twoheadrightarrow_\beta Z)\&(N \twoheadrightarrow_\beta Z)]$

3.6 Le λ-teorie

Una teoria è un insieme chiuso di equazioni cioè ogni equazione validabile tramite gli assiomi che definiscono la teoria fa parte della teoria stessa. Una teoria stabilisce i criteri in base ai quali due termini vengono considerati semanticamente uguali, ovvero dotati dello stesso comportamento operazionale.

[6] Analogamente alla forma β-normale, un termine M è detto in *R-normalizzabile* se non esiste alcun termine N tale che $M \to_R N$.

3.6.1 La teoria $\lambda\beta$

La teoria più debole che può essere definita su Λ viene chiamata $\boldsymbol{\lambda\beta}$-teoria. In tale teoria si considerano indistinguibili tutti i termini che sono β-congruenti. Le altre teorie sono estensioni del sistema assiomatico di $\boldsymbol{\lambda\beta}$.

Definizione 3.6.1 *I seguenti assiomi definiscono la* $\boldsymbol{\lambda\beta}$*-teoria:*

(i) $(\lambda x.M)\ N = M[x := N]$ $\quad \beta - congruenza$;

(ii) $(\lambda x.M) = (\lambda y.M)[x := y], con\ y \notin FV(M)$ $\quad \alpha - conversione$;

(iii)
$$\begin{array}{ll} M = M & prop.\ simmetrica; \\ M = N \Rightarrow N = M & prop.\ riflessiva; \\ M = N, N = P \Rightarrow M = P & prop.\ transitiva; \end{array}$$

(iv)
$$\begin{array}{ll} M = N \Rightarrow MP = NP & prop.\ applicazione\ 1; \\ M = N \Rightarrow PM = PN & prop.\ applicazione\ 2; \\ M = N \Rightarrow \lambda x.M = \lambda x.N & regola\ \xi. \end{array}$$

Definizione 3.6.2 *Due termini M e N si dicono* convertibili *e si scrive* $\lambda\beta \vdash M = N$ *se la loro eguaglianza è provabile a partire dagli assiomi della* $\boldsymbol{\lambda\beta}$*-teoria.*

Siano dati $M, N \in \Lambda$. Si noti che scrivere $\lambda\beta \vdash M = N$ corrisponde a dire che M è β-congruente a N ovvero $M =_\beta N$.

Un'aspetto fondamentale delle teorie, che è utile per stabilirne la sensatezza, è il concetto di *consistenza*.

Definizione 3.6.3

(i) Un'equazione è una formula del tipo $M = N$ *con* $M, N \in \Lambda$ *e viene detta* chiusa *se* $M, N \in \Lambda^0$*;*[7]

(ii) Una teoria $\mathcal{T}$ *si dice* consistente *e si scrive* $Con(\mathcal{T})$ *se esistono due termini la cui eguaglianza non può essere provata a partire dagli assiomi della teoria stessa;*

[7] Λ^0 è l'insieme dei λ-termini chiusi, cioè $M \in \Lambda^0 \Rightarrow FV(M) = \{\}$.

(iii) Una teoria $\mathcal{T}$ si dice inconsistente se non è consistente;

(iv) Sia $\mathcal{T}$ un insieme di equazioni. Allora si indica con $\lambda + \mathcal{T}$ la teoria ottenuta aggiungendo gli assiomi contenuti in $\mathcal{T}$ a quelli di $\boldsymbol{\lambda\beta}$;

(v) Si dice che $\mathcal{T}$ è consistente e si scrive $Con(\mathcal{T})$ se $Con(\lambda + \mathcal{T})$.

Proposizione 3.6.4 *La teoria $\boldsymbol{\lambda\beta}$ è consistente.*

L'idea intuitiva della dimostrazione di tale proposizione è di considerare due termini $M, N \in \Lambda$ che ammettano forma β-normale distinta. Se tali termini fossero eguagliabili tramite gli assiomi della $\boldsymbol{\lambda\beta}$-teoria essi sarebbero β-equivalenti. Ma, poiché $\twoheadrightarrow_\beta$ gode della proprietà di Church-Rosser , esisterebbe un ridotto comune delle due forme β-normali e questo creerebbe una contraddizione.

Per poter fare altri esempi di λ-teorie introduciamo il concetto di λ-termine *solvable* e *unsolvable*.

Definizione 3.6.5

(i) Sia $M \in \Lambda^0$. Allora M si dice solvable *se*

$$\exists n, \exists N_1 \cdots N_n \in \Lambda \qquad M N_1 \cdots N_n = \mathbf{I};$$

(ii) Un arbitrario $M \in \Lambda$ è solvable se la chiusura[8] di M, cioè $\lambda\vec{x}.M$ con $\{\vec{x}\} = FV(M)$, è solvable;

(iii) $M \in \Lambda$ è unsolvable *se non è solvable.*

Sia $\mathcal{T}$ un insieme di equazioni chiuse. Allora:

(i) $\mathcal{T}^+$ è l'insieme delle equazioni chiuse provabili in $\lambda + \mathcal{T}$;

(ii) $\mathcal{T}$ è una $\boldsymbol{\lambda}$-teoria se $\mathcal{T}$ è consistente e $\mathcal{T}^+ = \mathcal{T}$.

Ne deriviamo che ogni teoria è identificabile con l'insieme delle equazioni chiuse che sono provabili in essa.

[8]Sia $M \in \Lambda$ tale che $FV(M) = \{x_1, \ldots x_n\}$. Si dice chiusura di M il termine $\lambda x_1, \ldots, x_n.M$ e lo indicheremo con $\lambda\vec{x}.M$.

3.6.2 La teoria $\mathcal{H}$

Un classico esempio di λ-teoria è quella che considera indistinguibili tutti i termini unsolvable:

$$\mathcal{H}_0 = \{M = N | M, N \in \Lambda^0, unsolvable\}$$

$$\mathcal{H} = \mathcal{H}_0^+.$$

Il motivo per il quale si è pensato di definire tale teoria è che tutti i termini unsolvable sono privi di contenuto informativo. La teoria $\mathcal{H}$, quindi, non solo non crea problemi di incompatibilità fra λ-termini ma è molto utile per lo studio del λ-calcolo.

3.6.3 La teoria $\lambda\eta$

Uno degli aspetti che caratterizzano il λ-calcolo è la sua non estensionalità (altresì detta *intensionalità*). In un formalismo estensionale due termini sono uguali se hanno lo stesso comportamento funzionale, cioè se applicando gli stessi argomenti si ottengono gli stessi risultati. Due termini, invece, si dicono uguali dal punto di vista intensionale se individuano lo stesso processo di computazione.

Consideriamo, ad esempio, un λ-termine M. Poiché il λ-calcolo è non estensionale, M e $\lambda x.Mx$ sono diversi anche se il loro comportamento funzionale è il medesimo. Definiamo allora la teoria $\boldsymbol{\lambda\eta}$ nella quale considereremo indistinguibili i λ-termini che hanno comportamento computazionale identico. Tale teoria sarà, dunque, estensionale.

Definizione 3.6.6

(i) Regola di estensionalità(denotata con ***ext****):*

$$\forall M, N \in \Lambda \qquad Mx = Nx \;\Rightarrow\; M = N;$$

(ii) Denoteremo la teoria $\boldsymbol{\lambda\beta}$ *estesa con la regola* ***ext*** *con* $\boldsymbol{\lambda} + \boldsymbol{ext}$.

Per dotare il λ-calcolo della nozione di estensionalità è possibile, in alternativa, introdurre un nuovo assioma. La cosiddetta $\boldsymbol{\eta}$-conversione:

$$\forall M \in \Lambda \qquad \lambda x.Mx = M \quad (\text{con } x \notin FV(M)).$$

La teoria che aggiunge l' $\boldsymbol{\eta}$-conversione agli assiomi di $\boldsymbol{\lambda\beta}$ è denotata con $\boldsymbol{\lambda\eta}$.

Osservazione 3.6.7 *É possibile dimostrare che le teorie* $\boldsymbol{\lambda} + \boldsymbol{ext}$ *e* $\boldsymbol{\lambda\eta}$ *si equivalgono.*

3.7 Numerali di Church

Per comparare il λ-calcolo con gli altri formalismi computazionali dobbiamo trovare un metodo per rappresentare in esso i numeri naturali. A tal scopo introduciamo i numerali di Church.

Definizione 3.7.1

(i) per ogni $n \in \mathbb{N}$ e $F, A \in \Lambda$ definiamo $F^n(A)$ (applicazione iterata n volte di F su A) come segue:

$$\begin{array}{lcl} F^0(A) & = & A \\ F^{n+1}(A) & = & F(F^n(A)) \end{array}$$

(ii) Per ogni $n \in \mathbb{N}$, il numerale di Church c_n *è il λ-termine*

$$c_n = \lambda s.\lambda z.s^n(z);$$

e c_n è il numero n rappresentato nel formalismo del λ-calcolo.

I seguenti λ-termini sono numerali di Church:

(i) $c_0 = \lambda s.\lambda z.z$;

(ii) $c_1 = \lambda s.\lambda z.s\ z$;

(iii) $c_2 = \lambda s.\lambda z.s\ (s\ z)$;

(iv) $c_3 = \lambda s.\lambda z.s\ (s\ (s\ z))$;

(v) $c_4 = \lambda s.\lambda z.s\ (s\ (s\ (s\ z)))$;

3.8 Potere espressivo del λ-calcolo

Ogni λ-termine individua in modo naturale una funzione intesa in senso insiemistico. Più precisamente:

Sia dato un λ-termine $M \in \Lambda$. La funzione di arietà k *indotta* da M è la funzione $f : \Omega \longrightarrow \mathbb{N}$ tale che:

(i) $\Omega = \{(y_1, \ldots, y_k) \in \mathbb{N}^k \mid Mc_{y_1} \ldots c_{y_k} =_\beta c_i$ numerale di Church per un qualche $i\}$

(ii) per ogni $(y_1, \ldots, y_k) \in \Omega$, definiamo $f(y_1, \ldots, y_k) = i$ sse $Mc_{y_1} \ldots c_{y_k} =_\beta c_i$ per un certo i(per la proprietà di Church-Rosser tale c_i è unico).

Consideriamo un sottoinsieme Ω di $\mathbb{N}^k$ ed una funzione $f : \Omega \longrightarrow \mathbb{N}$. Si dice che $F \in \Lambda$ *rappresenta* f in λ-calcolo sse la funzione di arietà k indotta da F coincide con f.

Definizione 3.8.1 *Si dice che f è* rappresentabile *o λ-definibile sse esiste $F \in \Lambda$ che rappresenta f.*

Si noti che nel caso delle funzioni totali la definizione 3.8.1 si riconduce alla seguente e ben più nota definizione di λ-definibilità:

Definizione 3.8.2 *Si dice che una funzione numerica totale $f : \mathbb{N}^m \longrightarrow \mathbb{N}$ è* λ-definibile *se esiste un $F \in \Lambda$ tale che:*

$$Fc_{n_1} \ldots c_{n_m} =_\beta c_{f(n_1 \ldots n_m)}$$

per ogni $n_1 \ldots n_m \in \mathbb{N}$.

Proposizione 3.8.3

(i) Sia data una funzione ricorsiva $F \in \mathscr{R}$ con $Dim(F) = k$. Allora esiste un λ-termine $M \in \lambda$ che rappresenta F;

(ii) sia dato un λ-termine $M \in \Lambda$ allora esiste una funzione di arietà k indotta da M e tale funzione è rappresentabile nella teoria delle funzioni ricorsive.

Possiamo, dunque, dedurre che il potere espressivo del λ-calcolo è equivalente a quello della teoria delle funzioni ricorsive che, come avevamo visto nella osservazione 2.2.3 a pagina 14, individuava la classe $\mathscr{C}$ delle funzioni Turing-computabili.

Capitolo 4

Logica Combinatoria

Negli anni '20 Schönfinkel e Curry, poco prima che Church introducesse il λ-calcolo, inventarono un altro formalismo per rappresentare le funzioni detto *Logica Combinatoria*. Il nome "Logica Combinatoria", che viene sovente abbreviato in LC, deriva dal fatto che le funzioni in questo formalismo sono chiamate *funzioni combinatorie* o *combinatori*. Questo sistema si basa sull'idea di introdurre un nuovo simbolo ogni qual volta vi è la necessità di rappresentare una nuova funzione e di porre l'eguaglianza sintattica fra i due elementi. Questo modo di procedere può risultare scomodo, viene infatti naturale chiedersi se non sia possibile, partendo da un numero finito di funzioni, esprimere tutte le altre in funzione di queste ultime. Schönfinkel e Curry dimostrarono che fare ciò è possibile e che sono sufficienti due funzioni combinatorie dette, come vedremo poi, **K** e **S**. Un aspetto importante per cui il λ-calcolo differisce dalla Logica Combinatoria è che, in quest'ultima, non esiste il concetto di variabile legata.

4.1 Introduzione alla Logica Combinatoria

Gli oggetti di studio nella Logica Combinatoria sono i *termini combinatori.*

Definizione 4.1.1 *Siano dati:*

$$V = \{v_0, v_1, \ldots\}$$

un insieme che denota un alfabeto infinito di variabili, i simboli **K** *e* **S** costanti*, il simbolo* $=$ *detto* eguaglianza *e i simboli (e) che consentono la* parentesizzazione*. Viene dunque chiamato* C *l'insieme dei* termini combinatori *cioè l'insieme di tutte le stringhe definite tramite la seguente grammatica espressa in BNF:*

$$C ::= V|\mathbf{K}|\mathbf{S}|(CC)$$

con l'accortezza di considerare le stesse convenzioni notazionali concernenti le parentesi che abbiamo introdotto nel λ*-calcolo.*

In realtà l'alfabeto dei termini combinatori comprende anche il simbolo $\cdot$ che è l'operatore di applicazione ma, in favore della brevità, si usa scrivere ab in luogo di $a \cdot b$ e quindi non è stato introdotto. Supponiamo, inoltre, convenzionalmente che l'applicazione sia associativa a sinistra.

4.1.1 Riduzione

Poiché nella logica combinatoria i termini combinatori rappresentano delle funzioni dobbiamo definire, analogamente a quanto fatto nel formalismo del λ-calcolo la nozione di *riduzione* che ci permette di svolgere delle computazioni.

Definizione 4.1.2 *La relazione* $\rightarrow_w$ *di riduzione per i termini combinatori, chiamata* riduzione debole *è definita tramite le seguenti regole:*

- $\mathbf{K}AB \rightarrow_w A$*;*
- $\mathbf{S}ABC \rightarrow_w AC(BC)$*;*
- *se* $F \rightarrow_w F'$ *allora* $FG \rightarrow_w F'G$ *e* $GF \rightarrow_w GF'$*.*

Si noti che le regole in 4.1.2 determinano il comportamento operazionale dei combinatori **K** ed **S**. **K**, altresì detto *cancellatore*, si comporta come una funzione di arietà due che esegue la proiezione del primo dei due elementi che gli vengono applicati, mentre **S** necessita di tre argomenti, ad esempio a, b, c e li combina restituendo ac(bc).

Teorema 4.1.3 *Principio di Completezza Combinatoria: Qualsiasi funzione sia definita con l'operatore di applicazione è definibile in termini di* **K** *ed* **S**.

La dimostrazione del Teorema 4.1.3 si trova a pagina 40 di questa tesi.

Definizione 4.1.4

(i) Il simbolo $\twoheadrightarrow_w$ *denota la più piccola relazione riflessiva e transitiva contenente* $\rightarrow_w$*;*

(ii) Il simbolo $=_w$ *denota la minima relazione di equivalenza contenente* $\rightarrow_w$*;*

(iii) Un termine combinatorio F si dice in forma w-normale se non esiste un combinatore G tale che $F \rightarrow_w G$.

Ecco alcuni esempi:

(i) $\mathbf{I} \equiv \mathbf{SKK}$ si comporta come la funzione identica, infatti per ogni A si ottiene $\mathbf{I}A \rightarrow_w \mathbf{K}A(\mathbf{K}A) \rightarrow_w$ A;

(ii) **SII**(**SII**) si riduce a sè stesso;

(iii) Sia $\mathbf{W} \equiv \mathbf{SS}(\mathbf{KI})$, allora per ogni A, B si ottiene $\mathbf{W}AB \twoheadrightarrow_w ABB$;

(iv) $\mathbf{K}, \mathbf{S}, \mathbf{S}\ \mathbf{K}, \mathbf{K}\ \mathbf{S}$ *e* $\mathbf{S}\ \mathbf{K}\ \mathbf{K}$ sono in forma w-normale.

Si osservi che, come nel λ-calcolo i termini combinatori sono, al contempo, funzioni e argomenti.

4.1.2 Sostituzione

Analogamente a quanto fatto per il λ-calcolo introdurremo anche per i termini combinatori le nozioni di *sostituzione* e *variabili libere*.

Definizione 4.1.5 *Sia $F \in C$, allora definiamo l'insieme $FV(F)$ delle* variabili libere *di F nel seguente modo:*

$$\begin{array}{lcl} FV(x) & = & x; \\ FV(M\ N) & = & FV(M) \cup FV(N); \\ FV(\mathbf{K}) & = & \{\}; \\ FV(\mathbf{S}) & = & \{\}. \end{array}$$

Definizione 4.1.6 *Siano $M, N \in C$ e $x \in V$, allora definiamo induttivamente la* sostituzione *di x con N in M in tal modo:*

$$\begin{array}{lcl} x\{x := N\} & = & N; \\ y\{x := N\} & = & y \text{ se } x \neq y; \\ (H\ E)\{x := N\} & = & H\{x := N\}\ E\{x := N\}; \\ \mathbf{S}\{x := N\} & = & \mathbf{S}; \\ \mathbf{K}\{x := N\} & = & \mathbf{K}. \end{array}$$

Si noti che in Logica Combinatoria non esiste il concetto di *variabile legata* e non è necessario effettuare operazioni di rinomina nelle sostituzioni.

4.2 Le teorie della logica combinatoria

Ricordando che una teoria è un insieme chiuso di equazioni ci apprestiamo a definire una delle teorie della logica combinatoria.

4.2.1 La teoria CL

La teoria più debole che può essere definita sulla logica combinatoria viene chiamata *CL-teoria*.

Definizione 4.2.1 *I seguenti assiomi definiscono la* teoria CL*:*

$$
\begin{array}{lll}
(i) & \mathbf{K}AB = A & K; \\
(ii) & \mathbf{S}ABC = AC(BC) & S; \\
(iii) & \begin{array}{l} A = A \\ A = B \Rightarrow B = A \\ A = B, B = C \Rightarrow A = C \\ A = A', B = B' \Rightarrow AB = A'B' \end{array} & \begin{array}{l} \textit{prop. simmetrica}; \\ \textit{prop. riflessiva}; \\ \textit{prop. transitiva}; \\ \textit{applicazione}. \end{array}
\end{array}
$$

Definizione 4.2.2 *Dati due termini* A *e* B*, si scrive* $A =_{CL} B$ *se la loro eguaglianza è provabile a partire dagli assiomi della CL teoria cioè* $CL \vdash A = B$.

Dopo aver visto i concetti inerenti alla Logica Combinatoria vediamo come possono essere sfruttati per creare delle Strutture Applicative al fine di poter definire, nel prossimo capitolo, due funzioni che traducono i termini dalle λ-teorie alle algebre combinatorie e viceversa.

4.2.2 Algebre Combinatorie

Definizione 4.2.3 *Una* Struttura Applicativa *è una coppia* $(A, \cdot)$*, in cui* A *è un insieme arbitrario e* $\cdot$ *è un'operazione binaria su* A*, chiamata applicazione.*

Definizione 4.2.4 *Una struttura applicativa* $(A, \cdot)$ *si dice* estensionale *se dati* $a, b \in A$ *vale:*

$$\forall x \in A \quad (ax = bx) \Rightarrow a = b$$

Definizione 4.2.5 *Una* Algebra Combinatoria*(AC) è una struttura* $\mathrm{A} = (A, \cdot, \mathbf{K}, \mathbf{S})$*, in cui* $(A, \cdot)$ *è una Struttura Applicativa e* $\mathbf{K}$ *e* $\mathbf{S}$ *sono due elementi distinti di* A *che soddisfano le seguenti condizioni cioè per ogni* $a, b, c \in A$*:*

- $\mathbf{K}ab = a$;
- $\mathbf{S}abc = ac(bc)$.

Sia A una algebra combinatoria e M, N due termini. Scriveremo $A \models M = N$ se le interpretazioni di M ed N in A sono uguali.

Capitolo 5

Sull'equivalenza fra il λ-calcolo e la Logica Combinatoria

Una volta introdotto il sistema assiomatico del λ-calcolo e quello della Logica Combinatoria viene naturale interrogarsi sul fatto che essi siano, o meno, equivalenti.

5.1 Il λ-calcolo e la Logica Combinatoria

Cerchiamo, quindi, un modo per tradurre un termine del λ-calcolo in uno operazionalmente equivalente della Logica Combinatoria e un altro per tradurre un termine della Logica Combinatoria in uno operazionalmente equivalente del λ-calcolo.

5.1.1 Il λ-astrattore derivato

Definiamo quindi le seguenti funzioni.

Definizione 5.1.1 *Dato* $\mathbf{I} \equiv \mathbf{SKK}$*, si definisca induttivamente il* lambda astrattore derivato *della logica combinatoria (e si scriva* λ^**), nel* *seguente modo:*

$$\lambda^*x.M = \begin{cases} \lambda^*x.x &= \mathbf{I}; \\ \lambda^*x.A &= \mathbf{K}A & \textit{se } x \notin FV(A); \\ \lambda^*x.AB &= \mathbf{S}(\lambda^*x.A)(\lambda^*x.B) & \textit{altrimenti.} \end{cases}$$

Definendo il λ-astrattore derivato λ^* otteniamo come conseguenza di aver dimostrato il Principio di Completezza Combinatoria che era stato enunciato in 4.1.3 a pagina 35 in quanto, per ogni termine combinatorio A con variabili $x_1, \ldots, x_n$, esiste un termine chiuso f tale che $A \equiv fx_1, \ldots x_n$. Tale termine si può trovare ponendo $f = \lambda^*x_1, \ldots, x_n.A$. Qualsiasi funzione λ-definibile può essere espressa, dunque, solo in termini di $\mathbf{K}$ ed $\mathbf{S}$.

Usando il λ-astrattore derivato della logica combinatoria, possiamo definire una funzione $(\cdot)_{CL} : \Lambda \to C$ che traduca i termini combinatori in λ-termini, come segue:

$$\begin{aligned} x_{CL} &= x, \textit{ per } x \in V; \\ (M\ N)_{CL} &= M_{CL}N_{CL}; \\ \lambda x.M_{CL} &= \lambda^*x.(M)_{CL}. \end{aligned}$$

Possiamo definire, specularmente, una funzione $(\cdot)_\lambda : C \to \Lambda$ che traduca i λ-termini in termini combinatori, come segue:

$$\begin{aligned} x_\lambda &= x, \textit{ per } x \in V; \\ \mathbf{K}_\lambda &= \lambda xy.x; \\ \mathbf{S}_\lambda &= \lambda xyz.xz(yz); \\ (F\ G)_\lambda &= F_\lambda G_\lambda; \end{aligned}$$

Notazione 5.1.2 *Sia dato un termine combinatorio A, scriveremo $A_{\lambda,CL}$ al posto di $(A_\lambda)_{CL}$. Viceversa, sia dato un λ-termine M, scriveremo $M_{CL,\lambda}$ al posto di $(M_{CL})_\lambda$.*

5.2 Problematiche concernenti la traduzione

Essendo riusciti a determinare due funzioni, $(\cdot)_\lambda$ e $(\cdot)_{CL}$ che traducono i termini della logica combinatoria in termini del λ-calcolo e viceversa potremmo essere indotti a credere che i due sistemi siano del tutto equivalenti. In realtà ciò non è vero. Consideriamo, infatti, il combinatore:

$$\mathbf{K}x$$

che viene tradotto in λ-calcolo in:

$$(\mathbf{K}x)_\lambda = (\mathbf{K}x)_\lambda x_\lambda = (\lambda xy.x)x$$

è immediato notare che, mentre in logica combinatoria $\mathbf{K}x$ è una forma normale, la sua traduzione in λ-calcolo non lo è, è infatti possibile svolgere la seguente β-riduzione:

$$(\lambda xy.x)x \to_\beta \lambda y.x$$

Fra i due formalismi, dunque, il λ-calcolo è leggermente più espressivo come reso noto dal seguente lemma.

Lemma 5.2.1

(i) per ogni λ-termine M vale la regola $M_{CL,\lambda} =_\beta M$

(ii) per ogni A, B, termini combinatori vale $CL \vdash A = B \Rightarrow \lambda\beta \vdash A_\lambda = (B)_\lambda$.

(iii) per ogni M, N, λ-termini se vale $CL \vdash M_{CL} = N_{CL}$ allora $M =_\beta N$.

Non vale il viceversa, cioè siano dati A, B, termini della logica combinatoria allora:

$$\lambda\beta \vdash A = B \nRightarrow A_{CL} = B_{CL}$$

come si può notare dal contro-esempio che segue:

Esempio 5.2.2 *Consideriamo il termine combinatore* **SK***, allora*

$$(\mathbf{SK})_\lambda = \lambda yz.kz(yz) =_\beta \lambda y.zz$$

ma:

$$(\lambda y.zz)_{CL} = \mathbf{KI} = \mathbf{SKK}$$

e $CL \vdash \mathbf{SK} \neq \mathbf{SKK}$.

Inoltre, mentre è sempre vero che per ogni λ-termine M vale la regola $M_{CL,\lambda} =_\beta M$, come precedentemente enunciato nel Lemma 5.2.1, esiste almeno un termine combinatore A tale che $A_{\lambda,CL} \neq A$, come constatabile dall'esempio che segue.

Esempio 5.2.3 *Consideriamo il termine combinatore* $\mathbf{K}$ *e proviamo a calcolare* $\mathbf{K}_{\lambda,CL}$*:*

$$\begin{aligned} \mathbf{K}_{\lambda,CL} &= (\lambda xy.x)_{CL} \\ &= \lambda^* x.(\lambda y.x)_{CL} \\ &= \lambda^* x.(\lambda^* y.x) \\ &= \lambda^* x.\mathbf{K}x \\ &= \mathbf{S}(\lambda^* x.K)(\lambda^* x.x) \\ &= \mathbf{S}(\mathbf{KK})\mathbf{I} \end{aligned}$$

si noti che $\mathbf{K} \neq \mathbf{S}(\mathbf{KK})\mathbf{I}$ *quindi* $(\mathbf{K})_{\lambda,CL} \neq_w \mathbf{K}$.

Per rendere equivalenti il formalismo del λ-calcolo a quello della logica combinatoria è sufficiente *forzare* un numero finito di eguaglianze in quest'ultimo.

5.3 Le lambda algebre

Le algebre combinatorie equivalenti al λ-calcolo sono dette *λ-algebre* e vengono definite come segue.

Definizione 5.3.1 *Un algebra combinatoria A è chiamata* λ-algebra *se, per ogni A, B termini combinatori chiusi vale:*

$$\lambda\beta \vdash A_\lambda = B_\lambda \Rightarrow A \vdash A = B$$

Denoteremo la classe delle λ-algebre con LA.

Lemma 5.3.2 *LA è assiomatizzata dalle equazioni della teoria CL e dalle seguenti cinque equazioni chiuse dovute a Curry:*

(i) $\mathbf{K} = \mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})\mathbf{K}))(\mathbf{K}(\mathbf{SKK}))$

(ii) $\mathbf{S} = \mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{K}(\mathbf{S}(\mathbf{KS})))(\mathbf{S}(\mathbf{K}(\mathbf{S}(\mathbf{KK})))\mathbf{S})))(\mathbf{K}(\mathbf{K}(\mathbf{SKK})))$

(iii) $\mathbf{S}(\mathbf{KK}) = \mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})(\mathbf{S}(\mathbf{KS})\mathbf{K})))(\mathbf{KK})$

(iv) $\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})) = \mathbf{S}(\mathbf{KK})(\mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})(\mathbf{SKK})))(\mathbf{K}(\mathbf{SKK})))$

(v) $\mathbf{S}(\mathbf{K}(\mathbf{S}(\mathbf{KS})))(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KS}))) =$
$\mathbf{S}(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{KK})(\mathbf{S}(\mathbf{KS})(\mathbf{S}(\mathbf{K}(\mathbf{S}(\mathbf{KS})))\mathbf{S}))))(\mathbf{KS})$

Tale lemma, piuttosto controintuitivo, viene dimostrato formalmente in [Barendregt, 1984].

Siano dati M, N ctermini combinatori. Scriveremo $LA \vdash M = N$ oppure $M =_{LA} N$ se dagli assiomi del λ-calcolo arricchiti con quelli di Curry citati nel lemma 5.3.2 si può derivare che $M = N$.

Teorema 5.3.3

(i) Siano $M, N \in \Lambda$ allora:

$$M =_\beta N \text{ sse } LA \vdash M_{CL} = N_{CL};$$

(ii) Siano M, N combinatori allora:

$$LA \vdash M = N \text{ sse } \lambda\beta \vdash M_\lambda = N_\lambda;$$

(iii) In LA $M_{\lambda,CL} = M$.

Siano $M, N \in \Lambda$ tali che $M \equiv (\lambda x.x)x$ e $N \equiv x$. Allora:

$$M =_\beta N$$

cioè

$$(\lambda x.x)x =_\beta x$$

che è un eguaglianza valida, convertiamo entrambi i termini con $()_{CL}$

$$((\lambda x.x)x)_{CL} = \mathbf{I}x$$

$$(x)_{CL} = x$$

si ottiene $Ix =_{CL} x$ (vera).

$$\lambda x.M =_\beta \lambda x.N$$

cioè

$$\lambda x.(\lambda x.x)x =_\beta \lambda x.x$$

che è un eguaglianza valida, convertiamo entrambi i termini con $()_{CL}$

$$(\lambda x.(\lambda x.x)x)_{CL} = \mathbf{S}(\mathbf{KI})\mathbf{I}$$

$$(\lambda x.x)_{CL} = \mathbf{I}$$

si ottiene $\mathbf{I} \neq_{LC} \mathbf{S}(\mathbf{KI})\mathbf{I}$ mentre $\mathbf{I} =_{LA} \mathbf{S}(\mathbf{KI})\mathbf{I}$.

Teorema 5.3.4 *Esistono λ-algebre* **D** *tali che,*

$$\{M = N | (M, N \in \Lambda) \text{ e } (\mathbf{D} \models M_{CL} = N_{CL})\}$$

non formano una λ-teoria a causa del fallimento della regola ξ.

Per dimostrare il teorema 5.3.4 è sufficiente trovare una λ-algebra D, e due λ-termini M e N tali che:

$$D \models M_{CL} = N_{CL} \;\not\Rightarrow\; D \models (\lambda x.M)_{CL} = (\lambda x.N)_{CL}$$

Un esempio di λ-algebra di questo tipo lo si può trovare in [Barendregt, 1984].

5.4 I lambda modelli

Le λ-algebre in cui la regola ξ vale sono dette *λ-modelli* e sono definite come segue.

Definizione 5.4.1 *Sia* $\underline{C}$ *una λ-algebra in cui, dati M, N, combinatori polinomiali vale:*

$$\underline{C} \models M = N \Rightarrow \underline{C} \models \lambda^* x.M = \lambda^* x.N$$

allora $\underline{C}$ *è detta λ-modello.*

Lemma 5.4.2 *Se $\underline{C}$ è un λ-modello allora l'insieme $\{M = N | (M, N \in \Lambda) \ \& \ (\underline{C} \models M_{CL} = N_{CL})\}$ forma una λ-teoria.*

L'interrogativo che, a questo punto, viene natuale rivolgersi è se non sia possibile interpretare i λ-termini in un λ-modello $\underline{D} = (D, \cdot^D, \mathbf{K}^D, \mathbf{S}^D)$ senza dover passare obbligatoriamente per la loro traduzione in logica combinatoria cioè se $M \in \Lambda$ allora $|M|_D = |M|_{CL}$.

L'insieme D del λ-modello è diviso, chiaramente, in classi di equivalenza come mostrato dalla figura 5.1. Due elementi del λ-modello appartengono alla stessa classe di equivalenza sse il loro comportamento funzionale è il medesimo.

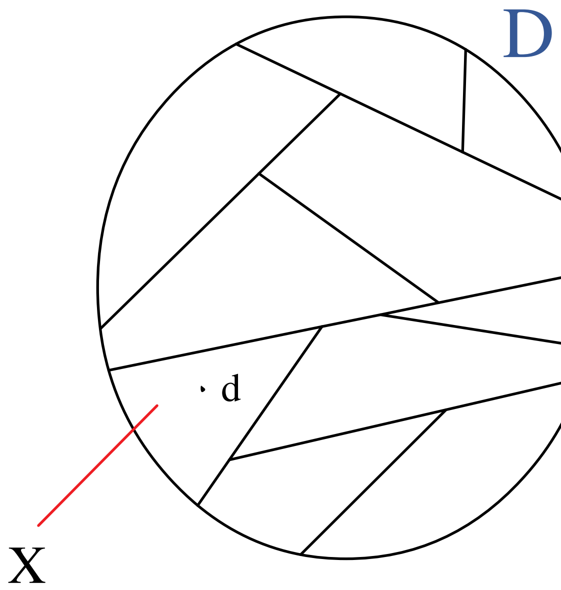

Figura 5.1: Classi di equivalenza dell'insieme D

Proviamo ad interpretare i λ-termini (estesi anche con le costanti) in $\underline{\text{D}}$ definendo la funzione $| \cdot |_{\underline{D}}$ in tal modo:

$$\begin{array}{lcl} |\overline{d}|_{\underline{D}} & = & d \in D \\ |M\ N|_{\underline{D}} & = & |M|_{\underline{D}} \cdot^D |N|_{\underline{D}} \\ |\lambda x.M|_{\underline{D}} & = & ? \end{array}$$

Nel definire $|\lambda x.M|_{\underline{D}}$ riscontriamo un problema: non abbiamo alcuna garanzia, infatti, sul fatto che la funzione $|\lambda x.M|_{\underline{D}}$ appartenga a D; e anche supponendo che $|\lambda x.M|_{\underline{D}} \in D$ sarà, comunque, all'interno di una classe di funzioni che hanno il suo stesso comportamento funzionale: in che modo è possibile scegliere un rappresentante?

Il λ-termine $|\lambda x.M|$ "rappresenta" la seguente funzione da D in D:

$$d \longmapsto |M[x := \overline{d}]|_{\underline{D}}$$

Quanti sono gli elementi di D che rappresentano questa funzione?

Poiché $\underline{D}$ è un modello:

$$X = \{e | e \cdot^D d = |M[x := overlined]|_D \text{ per ogni } d \in D\} \neq \emptyset$$

Ma come posso, all'interno di X, scegliere un rappresentante canonico?

Un possibile rappresentante è proprio $|\lambda^* x.M_{CL}|_{\underline{D}}$

Poiché $\underline{D} \models (\lambda^* x.(M)_{CL})\overline{d} = (M[x := \overline{d}])_{CL}$, la funzione:

$$|\lambda x.M|_{\underline{D}} \in X = \{\overline{d} \mid \overline{d} \cdot^D c = (M[x := \overline{c}])_{CL}\ per\ ogni\ c \in D\} \subseteq D$$

è un possibile rappresentante della classe X è proprio $|\lambda^* x.(M)_{CL}|_{\underline{D}}$.

Inoltre vale:

$$\underline{D} \models \overline{d}x = \overline{e}x \Rightarrow \lambda^* x.\overline{d}x = \lambda^* x.\overline{e}x\ per\ ogni\ \overline{d}, \overline{e} \in X$$

si ottiene:

$$\lambda x.\overline{d}x = (\lambda xy.xy)\overline{d}$$

e ponendo:

$$(\lambda xy.xy) \equiv \mathbf{1}$$

diventa:

$$(\lambda xy.xy)\overline{d} = \mathbf{1}\overline{d}$$

quindi possiamo definire:

$$|\lambda x.M|_{\underline{D}} = \mathbf{1}e \in C \text{ con } e \in X$$

.

Lemma 5.4.3 *Una λ-algebra $\underline{D}$ è un λ-modello se vale l'*assioma di Meyer-Scott *cioè per ogni $a, b \in D$,*

$$(\forall x)(ax = bx) \Rightarrow \mathbf{1}a = \mathbf{1}b$$

dove $\mathbf{1} \equiv \mathbf{SKI}$

Capitolo 6

Ulteriori approcci assiomatici alla Teoria della Computabilità

In questo capitolo introdurremo degli ulteriori approcci assiomatici alla Teoria della Computabilità. Essi sono importanti poiché, come vedremo, la classe di funzioni che identificano è la medesima delle funzioni Turing-computabili. Questo ci consente di trovare, conformemente alle esigenze e al problema da risolvere, modi diversi ma equivalenti per rappresentare le stesse funzioni e quindi di metterne in luce alcuni aspetti piuttosto che altri. Consideriamo, per primo, il formalismo della teoria classica della computabilità.

6.1 Computabilità Classica

In tale formalismo si utilizza una funzione di *codifica effettiva*[1] che associa un numero naturale ad ogni programma (inteso come stringa appartenente ad un qualche linguaggio). Chiaramente, questa operazione è resa possibile dal fatto che l'insieme di tutti i programmi ha la stessa cardinalità di $\mathbb{N}$. Senza perdita di generalità possiamo supporre che i programmi lavorino esclusivamente su dati naturali in quanto qualsiasi altro tipo di dato può essere codificato usando un numero naturale. Indicheremo con

[1] Una funzione si dice effettiva se è calcolabile e totale.

P_i il programma che viene codificato con il numero naturale i. Diremo, inoltre, che un programma P_i calcola la funzione $\Phi_i^k : \mathbb{N}^k \longrightarrow \mathbb{N}$ se[2]:

(i) $\Phi_i^k(x_1, \ldots, x_k) = r$ implica che il programma P_i, eseguito sull'input $(x_1, \ldots, x_k)$, termina con risultato r;

(ii) $\Phi_i^k(x_1, \ldots, x_k) = \uparrow$ implica che il programma P_i, eseguito sull'input $(x_1, \ldots, x_k)$, non termina l'esecuzione.

Si noti che la classe delle funzioni costruite in tal modo è chiusa rispetto la composizione, contiene tutte le funzioni di proiezione, contiene le funzioni S_n^m del *teorema s-m-n*[3] e contiene gli interpreti(che rappresentano l'applicazione generica).

6.1.1 Un'algebra combinatoria parziale definita sui naturali

Sia $A = (\mathbb{N}, \cdot, \mathbf{K}, \mathbf{S})$, in cui $\mathbb{N}$ è l'insieme dei numeri naturali, $\cdot$ è un operatore binario di applicazione e $\mathbf{K}, \mathbf{S}$ sono elementi distinti di $\mathbb{N}$. Possiamo definire:

(i) $\cdot : \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$ operazione binaria parziale:

$$a \cdot b \equiv \Phi_a(b);$$

(ii) un indice $\mathbf{K} \in \mathbb{N}$ tale che, per ogni $i, j \in \mathbb{N}$:

$$(\mathbf{K} \cdot i) \cdot j \simeq (\Phi_{\Phi_K(i)})(j) \simeq i$$

e $\mathbf{K} \cdot i \downarrow$. Scriveremo $a \cdot b \downarrow$ per indicare *$a \cdot b$ è definito* e $a \cdot b \uparrow$ per indicare *$a \cdot b$ è indefinito*, inoltre se t_1 e t_2 sono espressioni la scrittura $t_1 \simeq t_2$ abbrevia $(t_1 \downarrow \vee t_2 \downarrow) \Rightarrow t_1 = t_2$;

[2]Nel caso in cui sia chiara dal contesto la arietà delle funzioni Φ verrà omessa la k scritta in alto.

[3]Vedi Appendice A

(iii) un indice $\mathbf{S} \in \mathbb{N}$ tale che $\forall a, b, c \in \mathbb{N}$:

$$(((\mathbf{S} \cdot a) \cdot b) \cdot c) \simeq \Phi_{\Phi_{\Phi_{S(a)}(b)}}(c) \simeq (a \cdot c) \cdot (b \cdot c).$$

e $\mathbf{S} \cdot a \downarrow$, $(\mathbf{S} \cdot a) \cdot b \downarrow$.

Tralasciando il fatto che le funzioni definite come sopra sono parziali, possiamo notare che A corrisponde a un'algebra combinatoria come definita nella sezione 4.2.2 a pagina 37 di questo elaborato. Una struttura algebrica parziale definita sui numeri naturali come quella che abbiamo appena introdotto è in relazione con il formalismo classico della computabilità in quanto la classe delle funzioni rappresentabili in entrambi i formalismi è la medesima. Ma in che modo possono essere messe in correlazione tali entità? Come vedremo nei paragrafi 6.4.1 e 6.4.2 è possibile definire una corrispondenza fra Algebre Combinatorie Parziali e Strutture Applicative Effettive che sono, rispettivamente, la generalizzazione dell'algebra combinatoria parziale definita sui naturali e della teoria classica della computabilità. Con questa generalizzazione si intende effettuare un'operazione di astrazione su questi formalismi in modo tale che essi possano lavorare su un insieme arbitrario (e non necessariamente su $\mathbb{N}$).

6.2 Strutture Applicative Effettive

Le *Strutture Applicative Effettive* sono state introdotte e studiate da Asperti e Ciabattoni in [Asperti and Ciabattoni, 1995a, Asperti and Ciabattoni, 1997, Asperti and Ciabattoni, 1995b]). Come vedremo l'idea alla base delle Strutture Applicative Effettive è quella di rendere universale il formalismo della teoria classica della computabilità definendolo su un insieme generico. Tali strutture sono definite come segue.

Definizione 6.2.1 *Una* Struttura Applicativa Effettiva *(EAS) è una famiglia Φ^n di funzioni di ordine superiore*[4]*:*

$$\Phi^n : X \to \underbrace{(X^n \to X)}_{parziale} \quad n \in \mathbb{N}$$

[4] In accordo con la notazione classica della Teoria della Computabilità scriveremo Φ^n_i in luogo di $\Phi^n(i)$.

dove X è un insieme arbitrario e tale che i seguenti assiomi vengano soddisfatti:

(i) è chiuso rispetto alla composizione, cioè $\forall s, a_1, \ldots, a_r \in X$ *e* $\forall i, r \in \mathbb{N}, \exists f \in X$ *tale che:*

$$\Phi^i_f(x_1, \ldots, x_i) \simeq \Phi^r_s(\Phi^i_{a_1}(x_1, \ldots, x_i), \ldots, \Phi^i_{a_r}(x_1, \ldots, x_i))$$

$\forall (x_1, \ldots, x_i) \in X^i$*;*

(ii) contiene tutte le funzioni di proiezione I^n_i *cioè*

$\forall i, n \in \mathbb{N}, \exists k \in X$ *tale che:*

$$I^n_i \simeq \Phi^n_k$$

(iii) contiene le funzioni S^m_n *del* teorema s-m-n *della teoria classica della computabilità*[5]*, cioè*

$\forall m, n \in \mathbb{N}, \exists j \in X$ *tale che* Φ^{n+1}_j *è totale*[6] *e:*

$$\Phi^m_{\Phi^{n+1}_j(i, x_1, \ldots, x_n)}(y_1, \ldots, y_m) \simeq \Phi^{m+n}_i(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

con $(x_1, \ldots, x_n) \in X^n$ *e* $(y_1, \ldots, y_m) \in X^m$*;*

(iv) contiene gli interpreti *o* funzioni universali *che rappresentano l'applicazione generica, cioè*

$\forall r \in \mathbb{N}, \exists i$ *tale che:*

$$\Phi^{r+1}_i(x, y_1, \ldots, y_r) \simeq \Phi^r_x(y_1, \ldots, y_r)$$

$\forall x \in X,\ \forall (y_1, \ldots, y_r) \in X^r$.

Si noti che definendo una strutture applicativa effettiva con $X = \mathbb{N}$ ci si riconduce alla teoria classica della computabilità.

[5] Vedi Appendice A

[6] $\Phi^{n+1}_j = S^m_n$

6.2.1 Algebre Combinatorie Parziali

Come abbiamo precedentemente accennato le algebre combinatorie parziali possono essere viste come una generalizzazione dell'algebra parziale definita sui naturali introdotta nel paragrafo 6.1.1. Per poter formalizzare correttamente le algebre combinatorie parziali è utile definire prima le strutture applicative parziali.

Definizione 6.2.2 *Una* Struttura Applicativa Parziale *è una coppia* $(A, \cdot)$, *in cui* A *è un insieme arbitrario e* $\cdot$ *è un'operazione binaria parziale su* A*, chiamata applicazione.*

Anche in questo caso scriveremo $ab\downarrow$ per indicare *ab è definito* e $ab\uparrow$ per indicare *ab è indefinito*, inoltre se t_1 e t_2 sono espressioni applicative la scrittura $t_1 \simeq t_2$ abbrevia $(t_1\downarrow \vee t_2\downarrow) \Rightarrow t_1 = t_2$.

Definizione 6.2.3 *Una* Algebra Combinatoria Parziale*(PCA) è una struttura* $\mathrm{A} = (A, \cdot, \mathbf{K}, \mathbf{S})$*, in cui* $(A, \cdot)$ *è una Struttura Applicativa Parziale e* $\mathbf{K}$ *e* $\mathbf{S}$ *sono due elementi distinti di* A *che soddisfano le seguenti condizioni cioè per ogni* $a, b, c \in A$*:*

- $\mathbf{K}a\downarrow$, $\mathbf{S}a\downarrow, \mathbf{S}ab\downarrow$*;*
- $\mathbf{K}ab = a$*;*
- $\mathbf{S}abc \simeq ac(bc)$*.*

e $\cdot$ *è un'operazione binaria parziale su* A*, chiamata applicazione.*

Si noti che definendo una algebra combinatoria parziale con $A = \mathbb{N}$ ci si riconduce all'algebra parziale definita sui naturali.

In ogni PCA:

(i) $(ac(bc)\downarrow) \Rightarrow \mathbf{S}abc\downarrow$;

(ii) $(a\downarrow \mathbf{K}ab\uparrow) \Rightarrow b\uparrow$.

6.3 Un interessante modello di EAS derivante dal λ-calcolo

Un caso interessante di modello di Struttura Applicativa Effettiva si ottiene ponendo:

$$X \equiv \Lambda/\beta$$

cioè l'insieme dei λ-termini modulo la relazione di β-congruenza. In questa sezione scopriremo che dato:

$$\Phi^n_P(N_1, \ldots, N_n) = PN_1 \ldots N_n$$

l'insieme $\{\Phi^n\}_{n \in X}$ è un modello di Struttura Applicativa Effettiva. Infatti valgono le seguenti considerazioni:

(i) $\{\Phi^n\}$ soddisfa la chiusura rispetto la composizione, cioè
$\forall M_1, \ldots, M_l, \forall n \in \mathbb{N}$
$M_i \in X, \forall i \in \{1, \ldots, l\}$ si può definire un λ-termine in X tale che:

$$(\lambda y_1 \ldots y_l x_1 \ldots x_n . y_1(y_2 x_1 \cdots x_n) \cdots (y_l x_1 \cdots x_n)) M_1 \cdots M_l N_1 \cdots N_n =$$
$$M_1(M_2 N_1 \cdots N_n) \cdots (M_l N_1 \cdots N_n);$$

(ii) $\{\Phi^n\}$ contiene le proiezioni, cioè:

$$(\lambda x_1 \ldots x_k . x_i) N_1 \ldots N_k = N_i;$$

(iii) $\{\Phi^n\}$ contiene la funzione S^m_n del teorema s-m-n[7], cioè
$\forall m, n \in \mathbb{N}$ si può definire un λ-termine in X tale che:

$$(\lambda x y_1 \ldots y_n . x y_1 \ldots y_n) P N_1 \cdots N_n M_1 \cdots M_m = P N_1 \cdots N_n M_1 \cdots M_m;$$

(iv) $\{\Phi^n\}$ contiene gli interpreti, cioè
$\forall r \in \mathbb{N}$ si può definire un λ-termine in X tale che:

$$(\lambda x y_1 \ldots y_r . x y_1 \ldots y_r) M N_1 \cdots N_r = M N_1 \cdots N_r.$$

[7]Vedi Appendice A

Quindi, come volevamo dimostrare, l'insieme $\{\Phi^n\}_{n \in X}$ è un modello di Struttura Applicativa Effettiva.

Si noti che, per come sono state appena definite queste funzioni, il sistema che ne deriva è *totale*.

6.4 Equivalenza fra EAS e PCA

In questa sezione dimostreremo che ogni Struttura Applicativa Effettiva può essere messa in corrispondenza con un'Algebra Combinatoria Parziale e viceversa. Dalla *Turing Completezza*[8] delle Algebre Combinatorie Parziali potremo quindi inferire la Turing Completezza delle Strutture Applicative Effettive.

6.4.1 Da PCA a EAS

Sia $X = (X, \cdot, \mathbf{K}, \mathbf{S})$ un'Algebra Combinatoria Parziale come è stata definita in 6.2.3 a pagina 53. Allora definiamo una famiglia di funzioni parziali $\Phi^n : X \to (X^n \to X)$ *con* $n \in \mathbb{N}$ come segue:

$$\Phi_p^n(b_1, \ldots, b_n) \simeq pb_1 \ldots b_n$$

Notazione 6.4.1 *Nel caso in cui sia chiara dal contesto la arietà delle funzioni* Φ*, verrà omessa la* n *scritta in alto.*

Ricordiamo che la λ-astrazione può essere simulata nell'Algebra Combinatoria applicando le seguenti regole:

(i) $\lambda^* x.x \equiv \mathbf{SKK}$

(ii) $\lambda^* x.M \equiv \mathbf{K}M$ se $x \notin FV(M)$

(iii) $\lambda^* x.(M\ N) \equiv S(\lambda^* x.M)(\lambda^* x.N)$

Notiamo che:

[8] un modello si dice Turing completo se in tale modello sono rappresentabili tutte le funzioni Turing-computabili.

- l'insieme $\{\Phi^n\}$ è chiuso rispetto la composizione, infatti $\forall M_1 \dots M_l, \quad \forall n \in \mathbb{N}\ M_i \in X, \quad \forall i = 1 \dots l, \quad l \in \mathbb{N}, \exists F \in X$ tale che

  $$\Phi_{M_1}(\Phi_{M_2}(N_1 \dots N_n) \cdots \Phi_{M_l}(N_1 \dots N_n)) = \Phi_F(N_1 \dots N_n)$$

  con $F \equiv \lambda^* x_1, \dots, x_n.M_1(M_2 x_1 \dots x_n) \cdots (M_l x_1 \dots x_n)$;

- l'insieme $\{\Phi^n\}$ contiene le funzioni di proiezione, cioè

  $$\Phi^k_R(N_1, \dots, N_k) \simeq N_i$$

  con $R \equiv \lambda^* x_1, \dots, x_k.x_i$;

- l'insieme $\{\Phi^n\}$ contiene la funzione S^m_n del teorema s-m-n[9], cioè $\forall m, n \in \mathbb{N}, \exists Q \in X$ tale che $\Phi_Q(P, N_1, \dots, N_n)$ è definito e

  $$\Phi^m_{\Phi Q(P,N_1,\dots,N_n)}(M_1, \dots, M_m) \simeq \Phi^{m+n}_P(N_1, \dots, N_n, M_1, \dots, M_m)$$

  con $Q \equiv \lambda^* x_1, \dots, x_{n+1} y_1 \dots y_m.x_1, \dots, x_{n+1} y_1 \dots y_m$;

- l'insieme $\{\Phi^n\}$ contiene gli interpreti o funzioni universali, cioè $\forall n \in \mathbb{N} \exists U \in X$ tale che

  $$\Phi_U(M, N_1, \dots, N_n) \simeq \Phi_M(N_1, \dots, N_n)$$

  con $U \equiv \lambda^* x y_1, \dots y_n.x y_1, \dots, y_n$.

Allora $\Phi^n : X \to (X^n \to X)$ *con* $n \in \mathbb{N}$ è una Struttura Applicativa Effettiva.

6.4.2 Da EAS a PCA

Sia considerata una Struttura Applicativa Effettiva come definita in 6.2.1

$$\Phi^n : X \to (X^n \to X) \quad n \in \mathbb{N}$$

usiamo l'interprete della condizione (iv) della definizione 6.2.1 per definire un operazione binaria parziale:

[9]Vedi Appendice A.

$\cdot : X \times X \to X$

$$a \cdot b \equiv U^2(a,b) \equiv \Phi^2_u(a,b) \simeq \Phi_a(b)$$

(u è l'indice dell'interprete di arietà due) a questo punto **K** e **S** sono definiti come segue:

- *Dimostrazione dell'esistenza di* **K**

  Dimostriamo l'esistenza in X di un indice **K** tale che, per ogni $i, j \in X$

  $$(\mathbf{K} \cdot i) \cdot j \simeq (\Phi_k(i))(j) \simeq i$$

  infatti

  $$i \simeq I^2_1(i,j)$$

  dall'ipotesi *(ii)* di 6.2.1

  $$\simeq \Phi^2_l(i,j)$$

  $$\simeq \Phi^1_{S^1_1(l,i)}(j)$$

  dall'ipotesi *(iii)* di 6.2.1

  $$\simeq \Phi^1_{\Phi^2_m(l,i)}(j)$$

  $$\simeq \Phi^1_{\Phi^1_{S^1_1(m,l)}(i)}(j)$$

  poichè S^m_n è totale, ponendo $\mathbf{K} \equiv S^1_1(m,l)$, si ottiene

  $$\simeq \Phi^1_{\Phi^1_{\mathbf{K}}(i)}(j)$$

  allora, per come è stata definita la applicazione, diventa

  $$\simeq (\mathbf{K}(i))(j)$$

  cioè

  $$i \simeq \Phi^1_{\Phi^1_{\mathbf{K}}(i)}(j) \simeq (\mathbf{K}(i))(j)$$

  che è la definizione di **K**.

- *Dimostrazione dell'esistenza di* **S** Dimostriamo l'esistenza di un indice $\mathbf{S} \in X$ tale che $\forall a, b, c \in X$

$$(((\mathbf{S} \cdot a) \cdot b) \cdot c) \simeq \Phi_{\Phi_{\Phi_{S(a)}(b)}}(c) \simeq (ac)(bc)$$

infatti

$$(ac)(bc)$$

dalla definizione di applicazione, risulta

$$\simeq \Phi^2_z(\Phi^2_z(a, c), \Phi^2_z(bc))$$

cioè

$$\simeq \Phi^2_z(\Phi^2_z(I^3_1(a, b, c), I^3_3(a, b, c)), \Phi^2_z(I^3_2(a, b, c), I^3_3(a, b, c)))$$

dalle ipotesi *(i)* e *(ii)* di 6.2.1

$$\simeq \Phi^3_h(a, b, c)$$

$$\simeq \Phi^1_{S^2_1(h,a,b)}(c)$$

dall'ipotesi *(iii)* di 6.2.1

$$\simeq \Phi^1_{\Phi^2_j(h,a,b)}(c)$$

$$\simeq \Phi^1_{\Phi^1_{S^2_1(j,h,a)}(b)}(c)$$

dall'ipotesi *(iii)* di 6.2.1

$$\simeq \Phi^1_{\Phi^1_{\Phi^3_j(j,h,a)}(b)}(c)$$

$$\simeq \Phi^1_{\Phi^1_{\Phi^1_{S^2_1(j,j,h)}(a)}(b)}(c)$$

quindi la funzione S^m_n è totale e, ponendo $\mathbf{S} \equiv S^2_1(j, j, h)$ si ottiene

$$\equiv \Phi^1_{\Phi^1_{\Phi^1_{S}(a)}(b)}(c)$$

che, per definizione di applicazione è:

$$(a \cdot c) \cdot (b \cdot c) \simeq \Phi^1_{\Phi^1_{\Phi^1_{S}(a)}(b)}(c) \simeq (((\mathbf{S} \cdot a) \cdot b) \cdot c)$$

che è la definizione di $\mathbf{S}$.

Proposizione 6.4.2 *$\mathbf{K}x \downarrow$, $\mathbf{S}x \downarrow$ e per ogni $x, y \in X$ cioè S^m_n è una funzione totale.*

Osservazione 6.4.3 *Si può notare facilmente che* $\mathbf{K} = \mathbf{S} \Rightarrow |X| = 1$

Quindi, dati $X, \cdot, \mathbf{K}, \mathbf{S}$ definiti come sopra possiamo affermare che $X = (X, \cdot, \mathbf{K}, \mathbf{S})$ è una Algebra Combinatoria Parziale.

Avendo stabilito una *completa equivalenza* fra EAS e ACP possiamo dedurre che la classe di funzioni che i due sistemi individuano è la medesima cioè quella delle funzioni Turing Computabili.

6.5 Teoria delle Funzioni Ricorsive di Base

Il problema di definire un *approccio assiomatico* alla Teoria della Ricorsione è stato affrontato da molte persone che hanno proposto numerose soluzioni diverse. Qui di seguito presenteremo la Basic Recursive Function Theory di [Strong, 1968] che ha il pregio di essere strettamente collegata con le EAS appena introdotte.

6.5.1 BRFT

La differenza essenziale fra le EAS e la teoria delle funzioni ricorsive di base che stiamo per definire è che in questo sistema vengono aggiunte le costanti e le funzioni definite per casi[10].

[10] Si noti, comunque, che costanti e definizione per casi non sono essenziali per stabilire la Turing completezza del sistema.

Definizione 6.5.1 *Un BRFT*[11] *è una struttura* $(D, F, (\Phi^n)_{n\in\mathbb{N}})$ *che soddisfa le seguenti proprietà:*

- *D è un insieme infinito;*
- *F è un insieme di funzioni parziali in D tali che:*
  - *(i) è chiuso rispetto la composizione;*
  - *(ii) contiene tutte le funzioni di proiezione;*
  - *(iii) contiene la funzione per la definizione per casi cioè;*

$$f(x, a, b, c) = \begin{cases} b & se\ x = a \\ c & altrimenti \end{cases}$$

  - *(iv) contiene le funzioni S^m_n del* teorema s-m-n[12]*;*
  - *(v) contiene un interprete.*

Proposizione 6.5.2 *Tutte le funzioni ricorsive totali possono essere rappresentate in ogni BRFT.*

6.5.2 ω-BRFT

Un interessante esempio di BRFT viene chiamata ω-BRFT ed è definita come segue.

Definizione 6.5.3 *Un BRFT* $(\mathbb{N}, F, (\Phi^n)_{n\in\mathbb{N}})$ *in cui:*

(i) $\mathbb{N}$ *è l'insieme dei numeri naturali;*

(ii) F contiene la funzione successore;

si dice ω-BRFT

Lemma 6.5.4 *Se* $(\mathbb{N}, F, (\Phi^n)_{n\in\mathbb{N}})$ *è un ω-BRFT, allora F contiene tutte le funzioni ricorsive.*

[11] *Basic Recursive Function Theory*

[12] Vedi Appendice A

6.6 Strutture Uniformemente Riflessive

Un formalismo strettamente connesso alla teoria delle funzioni ricorsive di base che, come tale, va quantomeno menzionato e definito è quello delle strutture uniformemente riflessive introdotto da Strong in [Strong, 1968].

Definizione 6.6.1 *Un'algebra* $\mathbf{I} = (U, \cdot)$ *con dominio* U *e operazione binaria* $\cdot$ *è una* Struttura Uniformemente Riflessiva(URS) $\mathbf{I}$ *se, dati* α, ψ *e* $*$ *(che rappresenta l'*indefinito*) elementi di* U*, vengono soddisfatti i seguenti assiomi:*

(i) Assioma 1.

$$*a = a* = ** = *$$

(ii) Assioma 2.

$$\psi(x, a, b, c) = \begin{cases} b & \textit{se } x = a \\ c & \textit{altrimenti} \end{cases}$$

(iii) Assioma 3.

$$\alpha \neq \psi, \;\; \alpha ab \neq *, \;\; \alpha abc = ac(bc)$$

Ogni classe di funzioni che forma un modello di BRFT è a sua volta associata a una URS e viceversa (come dimostrato in [Strong, 1968]).

Capitolo 7

Conclusioni

Abbiamo, quindi, esaminato in modo più o meno schematico vari approcci assiomatici alla teoria della computabilità mettendone in evidenza le differenze e le corrispondenze cercando di avere sempre presente la classe di funzioni effettivamente rappresentabili tramite i suddetti formalismi. Nel caso del formalismo delle funzioni ricorsive parziali, che doveva fungere da punto di riferimento per mostrare la classe di funzioni effettivamente rappresentabili, è stata introdotta una sintassi *ad hoc* in quanto quella classica, ottima per scrivere funzioni ricorsive totali, mal si adatta per le funzioni ricorsive parziali. Sarebbe comunque interessante, in un qualche futuro, riuscire a dare una caratterizzazione formale della teoria assiomatica delle funzioni ricorsive parziali in cui venga separata la parte puramente sintattica da quella che denota la semantica operazionale di tali funzioni in quanto la letteratura scientifica sull'argomento non è del tutto esauriente, forse anche per il moderato utilizzo pratico di tali funzioni. Il metodo migliore, a mio avviso, sarebbe quello di scrivere una grammatica per generare tutte e sole le funzioni ricorsive parziali (lavoro reso difficoltoso dai problemi inerenti le differenti dimensionalità di queste funzioni) e solo in seguito specificarne il comportamento per consentire una più agile comprensione di tale formalismo.

Capitolo 8

Ringraziamenti

Vorrei, innanzitutto, ringraziare tutti coloro hanno permesso, o in qualche modo incoraggiato, sia la mia carriera scolastica che la stesura di questo elaborato.

I miei più sentiti ringraziamenti vanno, quindi, al Prof. Antonino Salibra per avermi iniziato alla Teoria della Computabilità e supportato in veste di relatore per questa tesi, agli i.SenSE che hanno aiutato la mia crescita professionale, ad Andrea Pavan il quale, sottoponendo alla mia attenzione problemi e paper a carattere Matematico fra i più variegati, fa sì che la mia mente non si crogioli mai nella nullafacenza, al Dott. Claudio Mirolo dell'Università di Udine per essere stato un punto di riferimento durante il mio primo anno di studi in ambito informatico e per avermi insegnato in modo semplice ed efficace l'arte del ragionamento ricorsivo ed, infine, al Dott. Alvise Spanò, al Dott. Giulio Marcon, al Dott. Diego Dainese e al Dott. Enrico Zeffiro per aver letto questa tesi e avermi dato un loro giudizio. Ma il grazie più grande e sincero va, forse, ai miei genitori che mi hanno supportato economicamente durante questi lunghi anni di studio e a tutti i miei amici che ormai da tempo mi circondano e con i quali ho avuto occasione di condividere tante esperienze formative. Grazie.

Bibliografia

[Ambriola and Mirolo, 1994] Ambriola, V. and Mirolo, C. (1994). *Programmazione Razionale*. Jackson Libri, first edition.

[Asperti and Ciabattoni, 1995a] Asperti, A. and Ciabattoni, A. (1995a). Effective applicative structures. In *Category Theory and Computer Science*, pages 81–95.

[Asperti and Ciabattoni, 1995b] Asperti, A. and Ciabattoni, A. (1995b). On completability of partial combinatory algebras.

[Asperti and Ciabattoni, 1997] Asperti, A. and Ciabattoni, A. (1997). A sufficient condition for completability of partial combinatory algebras. *The Journal of Symbolic Logic*, 62(4):1209–1214.

[Barendregt, 1984] Barendregt, H. P. (1984). *The Lambda calculus - It's Syntax and Semantics*. North-Holland, Amsterdam.

[Cutland, 1980] Cutland, N. J. (1980). *Computability: An Introduction to Recursive Function Theory*. Cambdridge University Press, first edition.

[E.Wagner, 1969a] E.Wagner (1969a). Uniformly reflexive structures: an axiomatic approch to computablity. *Information Sciences*, 1:343–362.

[E.Wagner, 1969b] E.Wagner (1969b). Uniformly reflexive structures: on the nature of godelizations and relative computability. *Trans. Americ. Math. Soc.*, 114:1–41.

[Hopcroft et al., 2001] Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2001). *Introduction to Automata Theory, Languages and Computation.* Low Price Edition. Addison Wesley Longman, Inc, Reading, Mass., USA, 2 edition.

[Manca., 1981] Manca., V. (1981). Computational formalism: abstract combinatory view point and related test order logical framework. *Fundamenta Informaticae*, 4:3–18.

[Schonfinkel, 1924] Schonfinkel, M. (1924). Uber die bausteine der mathematischen logik.

[Selinger, 1996] Selinger, P. (1996). The lambda calculus is algebraic.

[Sòrensen and Urzyczyn, 1998] Sòrensen, M. H. and Urzyczyn, P. (1998). Lectures on the Curry-Howard isomorphism. Technical report 98/14 (= TOPPS note D-368), Univ. of Copenhagen.

[Strong, 1968] Strong, H. R. (1968). Algebraically generalized recursive function theory. *IBM Journal of Research and Development*, 12(6):465–475.

Appendice A

Teorema s-m-n

Avendo più volte citato nel capitolo 6 di questo elaborato il *teorema s-m-n* altresì detto *teorema del parametro* mi sembra doveroso quantomeno scrivere il suo enunciato per dare la possibilità a chi non dovesse conoscerlo di capire di cosa si tratti. Per la dimostrazione si rimanda a [Cutland, 1980]. In accordo con la notazione classica della Teoria della Computabilità scriveremo Φ_i^n in luogo di $\Phi^n(i)$.

Teorema A.0.2 *Teorema s-m-n (o del parametro).*

Per ogni $m, n \geq 1$ *esiste una funzione* totale *e* computabile *di arietà* $(m+1)$ *che indicheremo con* $S_n^m(e, \overline{x})$ *tale che per ogni* $\overline{x}$*:*

$$\Phi_e^{m+n}(\overline{x}, \overline{y}) \simeq \Phi^n_{S_n^m(e,\overline{x})}(\overline{y})$$

Indice analitico